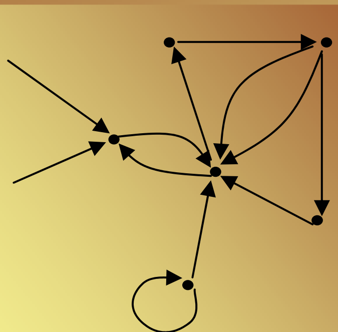
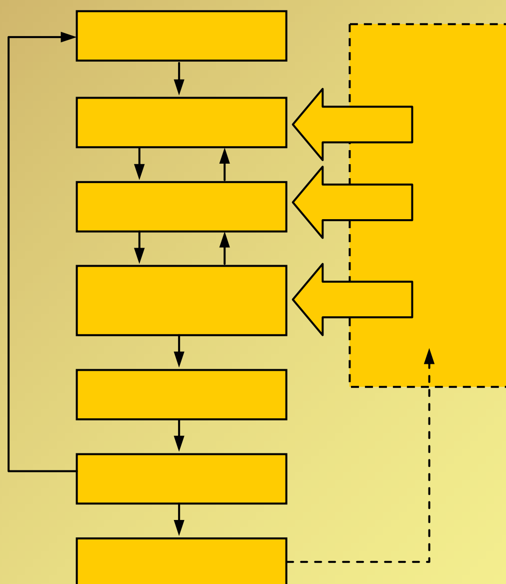
RODIAN SCÎNTEIE





BAZE DE DATE ŞI ALGORITMI PENTRU CĂI DE COMUNICAȚIE



EDITURA SOCIETĂȚII ACADEMICE "MATEI-TEIU BOTEZ"
IAŞI - 2003

RODIAN SCÎNTEIE

BAZE DE DATE ŞI ALGORITMI PENTRU CĂI DE COMUNICAȚIE

Editura Societății Academice ”Matei-Teiu Botez”

Iaşi - 2003

Referenți:
prof.univ.dr.ing. Gabriela VIORER
prof.univ.dr.ing. Cornel JIVA

Descrierea CIP a Bibliotecii Naționale a României
SCÎNTEIE, RODIAN
Baze de date şi algoritmi pentru căi de comunicații / Rodian Scînteie. - Iaşi : Editura Societății Academice "Matei - Teiu Botez", 2003
Bibliogr.
ISBN 973-86343-3-4

625.7(075.8)

CUPRINS

ISBN 973-86343-3-4 Editura Societăţii Academice „Matei-Teiu Botez”
Iaşi 2003

Capitolul 1

INTRODUCERE ÎN MANAGEMENTUL ASISTAT DE CALCULATOR

1.1. INTRODUCERE

Infrastructura transporturilor reprezintă o avuţie publică de o valoare imensă care trebuie transmisă într-o stare mulţumitoare către generaţiile viitoare. Responsabilitatea factorilor de decizie este cu atât mai mare cu cât de buna funcţionare a căilor de comunicaţie depinde desfăşurarea normală a întregii activităţi economice a ţării, securitatea colectivă şi, bineînţeles, siguranţa participanţilor la trafic.

Menţinerea în stare de funcţionare a infrastructurii depinde de modul în care administratorul reţelei efectuează acţiuni de intervenţie. Pentru ca aceste acţiuni să fie eficiente trebuie cunoscut în amănunt situaţia calitativă şi funcţională a componentelor. Pentru aceasta sunt necesare programe sistematice de inventariere, inspecţie şi evaluare. Scopul unor astfel de programe este de a stabili capacitatea actuală şi de perspectivă a infrastructurii de a face faţă cerinţelor şi scopului pentru care a fost realizată.

Pe baza datelor existente se pot stabili strategii, programe, proiecte, soluţii etc. Pentru realizarea lor sunt necesare fonduri substanţiale. Suma valorii tuturor proiectelor reprezintă bugetul necesar pentru administrare. Practica a demonstrat că fondurile disponibile nu sunt niciodată suficiente nicăieri în lume. Deoarece pentru aceleaşi fonduri concurează mai multe proiecte, administratorii trebuie să aleagă între acestea. Ei trebuie să aleagă între proiecte localizate în diverse situri dar şi între variante de proiect pentru aceeaşi locaţie.

Decizia este dificilă deoarece în cele mai multe cazuri este necesară predicția efectelor diferitelor proiecte. În final trebuie selectate acele proiecte care aduc cele mai mari beneficii. Prin beneficii trebui să înțelegem nu numai valoarea bănească ci şi de altă natură cum ar fi timp, consum de combustibil şi piese de schimb etc. Studiul sistematic al mecanismelor matematice şi logice de descriere a comportării rețelei şi utilizatorilor, combinat cu dezvoltarea şi utilizarea de algoritmi de predicție, poate ajuta la eliminarea sau întârzierea apariției unor fenomene negative sau se pot minimiza efectele lor încă înainte de apariție.

Simultan cu programele de întreținere a infrastructurii trebuie inițiate şi parcurse permanent programe de culegere de date pentru a putea cunoaşte starea şi evoluția sistemului. O cunoaştere mai profundă implică utilizarea unui număr cât mai mare de variabile de descriere. Numărul combinațiilor posibile devine astfel tot mai mare şi mintea umană poate tot mai greu să le cuprindă şi să le controleze integral. În consecință, se impune o gândire sistemică şi o abordare algoritmică. Complexitatea tot mai mare a algoritmilor şi dimensiunea crescândă a bazelor de date în care se stochează datele impun utilizarea calculatoarelor ca mijloc de lucru.

A fost conceput un cadrul general care include instrumente eficiente de evaluare, analiză economică, metode moderne de management. Termenul generic este management asistat de calculator. Scopul unui astfel de mecanism este de a obține un beneficiu maxim din utilizarea resurselor disponibile.

Managementul infrastructurii rutiere se confruntă cu o permanentă obligație de a lua decizii. Aceste decizii se referă la alocarea de resurse, fie ele materiale, financiare, umane sau de echipament. În contextual actual de desfăşurare a proceselor alocarea este irevocabilă.

Avem aici câteva noțiuni ce necesită o explicație. Irevocabil şi decizie vor fi reținuți cu înțelesul exprimat de R.A. Howard (1966) [12]:

> *„ ... irevocabil în sensul că este imposibil sau extrem de costisitor să revenim la situația care exista înainte de luarea deciziei. De aceea pentru scopul propus decizie nu este hotărârea mintală de a urma o anumită linie de acțiune ci mai degrabă ducerea la îndeplinire a acțiunii în conformitate cu hotărârea.”*

Procesul decizional este unul complex afectat frecvent de subiectivism şi de incertitudine. O dată cu pătrunderea calculatorului electronic în majoritatea domeniilor vieții şi managementul infrastructurii transporturilor rutiere trebuie să beneficieze de avantajele utilizării noilor tehnologii. Viteza de calcul deosebită, capacitatea de stocare a datelor şi procedurile sofisticate de prelucrare permit specialistului în inginerie civilă să facă analize pertinente, multicriteriale care să permită luarea de decizii corecte, documentate şi argumentate.

1.2. OBIECTUL CURSULUI

Acest curs va trata modul în care datele, informaţiile şi cunoştinţele pot fi obţinute, stocate, regăsite şi prelucrate pentru a fi utilizate în managementul infrastructurii transporturilor. Un accent deosebit se va pune pe metodele algoritmice de procesare cu scopul luării deciziilor.

În capitolele următoare vom face doar o trecere în revistă şi o introducere în mijloacele moderne de analiză care transpuse în programe rulate pe calculatoarele moderne pot oferi sprijin în procesul decizional al managementului infrastructurii transporturilor. Considerăm că este de o importanţă deosebită cunoaşterea unor noţiuni de bază privind:

- Definirea, reprezentarea şi gruparea datelor;
- Achiziţionarea, stocarea, regăsirea şi actualizarea datelor în baze de date;
- Prelucrarea datelor pe baza algoritmilor pentru obţinerea de informaţii;
- Aplicaţii ale utilizării bazelor de date şi a algoritmilor în managementul infrastructurii transporturilor;
- Modelarea şi simularea în vederea prelucrării datelor .

Studierea şi aprofundarea acestor noţiuni permite colaborarea facilă cu ceilalţi specialişti implicaţi în colective multidisciplinare care tratează probleme de trafic, administrarea infrastructurii, sisteme informaţionale integrate, dirijarea transporturilor, cercetări operaţionale etc.

Acest curs NU se va ocupa de: limbaje de programare, arhitectură hardware, arhitectură software sau de problematica proiectării şi implementării de programe de calcul. Acestea sunt doar tangenţial abordate. Chiar dacă nu este absolut necesară, cunoaşterea unui limbaj de programare va facilita buna înţelegere a noţiunilor cu care ne vom confrunta.

Acest curs NU se ocupă de teoria şi practica managementului cu aplicare în reţeaua infrastructurii transportului rutier. Este totuşi strâns interconectat, logica şi metodele de raţionament însuşite în cadrul acestui curs putând fi aplicate în rezolvarea nevoilor managementului.

1.3. DATE, INFORMAŢII, CUNOŞTINŢE

În managementul infrastructurii rutiere, de altfel ca în multe alte domenii şi ca în viaţa însăşi, există o activitate susţinută şi permanentă de culegere, prelucrare, generare şi transmitere a datelor, informaţiilor şi cunoştinţelor.

La nivelul elementar putem spune următoarele:

- Datele – reprezintă elemente simbolice de bază, cărămida în construcţia cunoaşterii;

- Informația – reprezintă datele prelucrate pentru a corespunde unui scop şi care răspunde la întrebările: Cine? Ce? Unde? Când? Cât?
- Cunoştințele – aplicația datelor şi informațiilor şi răspunde la întrebarea: Cum?
- Înțelegerea – apreciază: De ce?
- Înțelepciunea – apreciază înțelegerea?

Pentru a asigura o bună înțelegere a conceptelor cuprinse în această lucrare şi a desfăşurării proceselor decizionale care decurg la diferitele niveluri ale managementului infrastructurii transporturilor vom insista asupra noțiunilor de date, informații şi cunoştințe. Celelalte depăşesc limitele pe care ni le-am impus şi pot fi abordate într-o eventuală lucrare de filozofia cunoaşterii.

Informațiile şi cunoştințele se obțin din prelucrarea datelor după o schemă principială prezentată în figura următoare.

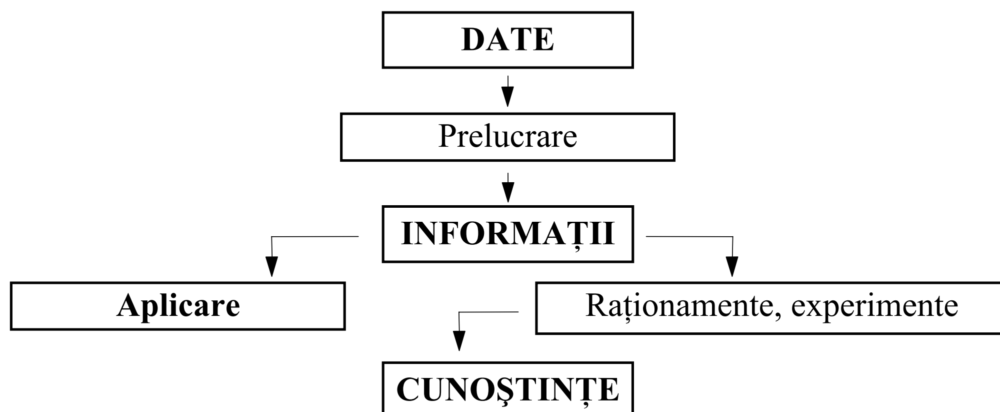


Fig. 1 Procesul de transformare a datelor în cunoştințe [13]

DATELE

Activitatea specialiştilor din domeniu este caracterizată prin entități factice exprimate fie sub formă de valori numerice, fie ca percepții şi observații numerice. Aceste entități faptice independente şi neevaluate, existente în număr mare, se numesc *date* [13].

Datele sunt o reprezentare a observațiilor, faptelor, conceptelor sau instrucțiunilor într-o manieră formală adaptată comunicării, procesării şi interpretării de către om sau cu mijloace automatizate.

Date sunt toate acele valori numerice, caracter, sau de altă natură ce pot fi păstrate separat sau grupate pentru a fi stocate, regăsite, transmise, prelucrate etc. pentru a obține informații.

Datele reprezintă elementul de bază, materia primă, în procesul decizional. Pot avea diferite reprezentări şi pot fi utilizabile sau nu. Ele pot exista în variate forme: numere, texte, mulţimi, colecţii etc., dar nu au semnificaţie prin ele însele.

Datele pot fi de patru tipuri (Floridi-1999 [10]):

- Date primare – datele brute culese din teren, stocate în bazele de date sau înscrise în cărţi, simple înşiruiri de valori care nu au suferit o prelucrare prealabilă sau procesul de prelucrare a fost sumar. Sistemele de gestiune a informaţiilor sunt proiectate, în fapt, să manipuleze în special astfel de date.
- Metadate – sunt date secundare care oferă indicaţii şi descriere privind natura datelor primare. Ajută sistemele de gestiune a bazelor de date să-şi îndeplinească sarcina prin descrierea proprietăţilor esenţiale ale datelor primare: localizare, format, actualizare, disponibilitate, drepturi de autor, etc.
- Date operaţionale – sistemele de gestiune a bazelor de date monitorizează şi colectează date privind propria funcţionare, despre funcţionarea sistemului de calcul sau a performanţelor implementării. Aceste date pot asigura feedback-ul, bucla de control, pentru identificarea şi corectarea eventualelor erori de funcţionare, a greşelilor de proiectare sau a neconcordanţelor cu necesităţile utilizatorului;
- Date derivate – sunt acele date care pot fi deduse sau extrase din tipurile anterioare când acestea sunt utilizate pentru comparaţii şi analize cantitative.

Ca toate produsele societăţii umane datele au o un ciclu de viaţă. Funcţie de durata lor de viaţă datele pot fi:

- Volatile: date care se obţin, se prelucrează, se utilizează în generarea informaţiilor sau altor tipuri de date şi a căror valoare nu se păstrează, sau
- Persistente: date stocate pe un suport de informaţie cum ar fi hârtie, benzi perforate, mijloace magnetice (benzi, dischete, discuri), mijloace optice (compact discuri) etc.

INFORMAŢIA

Datele obţinute în cadrul activităţilor de proiectare, execuţie, investigaţie, mentenanţă etc. constituie un material informaţional brut care poate fi ordonat şi prelucrat, având în vedere diferite obiective. În urma acestui proces de transformare a datelor, se obţin *informaţii*, care reprezintă o interpretare a datelor în conformitate cu anumite situaţii particulare sau cu înţelegerea de către mintea umană în general.

Informaţia este trecerea de la valoare la semnificaţia care se ataşează datei. Înţelesul care se ataşează datelor depinde de locul, modul şi scopul obţinerii lor. Această semnificaţie poate fi utilă dar acest lucru nu este obligatoriu.

Există numeroase încercări de a defini informaţia. Fiecare este adevărată pe domenii limitate dar nici una nu este suficient de generală. Studii de sinteză făcute de-a lungul timpului nu au găsit la autorii citaţi o unitate de vedere (Braman 1989 [3], Losee 1997 [15], Machlup 1983 [17], NATO 1974 [21], 1975 [22], 1983 [23], Schrader 1984 [25], Wellisch 1972 [32], Wersig şi Neveling 1975 [33]). Deşi pare o noţiune simplă, conceptul de informaţie este greu de cuprins într-o singură definiţie şi poate fi explicat în moduri diferite funcţie de categoria de cerinţe şi deziderate care orientează teoria (Bar-Hillel şi Carnap 1953 [1], Szaniawski 1984 [29], Shannon 1993 [27]).

Dintre definiţiile date şi comentariile făcute putem reţine:

- Informaţia reprezintă datele care au fost procesate într-o formă inteligibilă pentru receptor (Davis şi Olson 1985 [6]);
- Datele reprezintă materia primă care este procesată şi rafinată pentru a obţine informaţia (Silver şi Silver 1989 [28]);
- Informaţie egal date plus semnificaţie (Checkland şi Scholes 1990 [5]);
- Informaţia este setul de date care a fost interpretat şi înţeles de receptorul mesajului (Lucey 1991 [16]);
- Datele trebuie interpretate sau prelucrate pentru a deveni informaţie (Warner 1996 [31]).

Noţiunea de informaţie este adesea utilizată intuitiv pentru a desemna conţinutul semantic non-mental, independent de utilizator, declarativ inclus în implementări fizice precum baze de date, enciclopedii, locaţii internet, programe tv. (Buckland 1991 [4]), care pot fi colectate, accesate sau prelucrate (Floridi 2002 [9]).

Se pune problema conţinutului de adevăr a informaţiei. Care este situaţia entităţilor care sunt privite drept informaţie dar care în fapt nu conţin valoare de adevăr. Unii autori consideră că informaţia falsă este pseudo-informaţie (Floridi 2002 [9]). Alţi autori sunt şi mai categorici: „informaţia falsă nu este un tip inferior de informaţie; ea pur şi simplu nu este informaţie” (Grice 1989 [11]); „informaţia falsă sau informaţia greşită nu sunt tipuri de informaţie” (Dretske 1981 [8]).

Pornind de la cele subliniate mai sus şi pe baza abordării metodologice dezvoltate în logica situaţională de Barwise şi Perry 1983 [2]; Israel şi Perry 1990 [14]; Devlin 1991 [7], Luciano Floridi 2002 [9] dă următoarea definiţie semantică pentru elementul individual de informaţie:

σ este informaţie obiectivă, semantică dacă şi numai dacă:

- σ constă într-un set D nevid de date (d);
- datele din D sunt bine formatate;
- datele din D au semnificaţie;
- datele d din D sunt veridice.

S-a utilizat termenul „veridic” în loc de „adevărat” cu sensul de „reprezentând sau transmiţând un conţinut cu grad de adevăr despre o situaţie dată”. Toate discuţiile ulterioare şi referirile la noţiunea de informaţie din această lucrare ţin cont de această definiţie.

Informaţia reprezintă un element fundamental al activităţii în orice întreprindere (Păunescu et al. 1985 [24]). Ea prezintă caracteristici similare celor pe care le au bunurile materiale: se produce, se stochează şi se prelucrează, este perisabilă în timp şi are un preţ de cost – exprimat prin suma cheltuielilor ocazionate de obţinerea, prelucrarea, memorarea sau difuzarea ei.

Indicii de calitate ai informaţiei sunt: precizia, oportunitatea, completitudinea.

Precizia este definită prin cantitatea de informaţie corectă în raport cu întregul volum de informaţii produs într-o anumită perioadă de timp. Există un raport direct între acest indice de calitate şi datele pe care se fundamentează informaţiile respective; pentru a creşte valoarea acestui indice este necesar un volum mai mare de date. În acelaşi timp însă obţinerea informaţiilor dintr-o cantitate mare de date impune un volum mai mare de prelucrări care, în anumite situaţii, poate veni în contradicţie cu alt indice de calitate şi anume cu oportunitatea.

Relevanţa sau actualitatea exprimă faptul că o informaţie este utilă într-un anumit moment, legat de desfăşurarea în timp a unor fenomene. Obţinerea informaţiilor, care privesc aceste fenomene, după depăşirea unor etape ale evoluţiei lor reduc sau anulează valoarea acestor informaţii. Corelat cu precizia apare o anumită contradicţie, a cărei soluţionare depinde de natura şi specificul activităţii în cauză. Fenomenul de perisabilitate în timp a informaţiilor se face simţit prin diminuarea valorii informaţiei în raport cu preţul de obţinere al acestora.

Completitudinea exprimă necesitatea de a dispune de cât mai multe sau chiar de totalitatea informaţiilor referitoare la un domeniu al activităţii umane.

Obţinerea informaţiilor cu calităţile menţionate anterior este condiţionată atât de datele care le generează, cât şi de mijloacele de prelucrare disponibile. Dacă datele folosite în procesul de obţinere a informaţiilor sunt inexacte, incomplete sau perimate, informaţiile care rezultă sunt de valoare redusă. Orice informaţie trebuie să reducă din incertitudine şi să conducă la o mai bună înţelegere a anumitor situaţii sau fenomene.

Potrivit lui Zadeh (1997 [34]) informaţiile disponibile se pot grupa în trei categorii:

- Informaţii factice care sunt numerice şi bazate pe măsurători (ex. „înălţimea grinzii este de 82,5 cm”);
- Informaţii pseudo-numerice şi bazate pe pseudo-măsurători (ex. „ne întâlnim la trei”);
- Informaţii bazate pe percepţii care sunt în special aprecieri lingvistice (ex. „cineva este cinstit şi arătos”).

Aceste grupe diferă între ele prin gradul de incertitudine.

Când există incertitudine în aprecierea unui fenomen se poate defini cantitatea de informaţie care înlătură această incertitudine. Informaţia este în acest caz numită şi entropie şi este reprezentată de numărul minim de întrebări ce trebuie pus în medie pentru a elimina incertitudinea.

Numeric, informaţia poate fi estimată cantitativ funcţie de probabilitatea stărilor. Astfel, considerând un număr de stări: $x_1, x_2, ..., x_n$ a căror probabilitate de apariţie este dată de $P(x_i)$, cantitatea de informaţie se calculează cu formula:

$$I = -\sum_i P(x_i) \log_2 P(x_i) \tag{1}$$

Logaritmul în baza doi implică măsurarea informaţiei în biţi. Trebuie făcută convenţia $0 \log_2(0) = 0$, adică o stare cu probabilitatea 0 nu aduce informaţie în sistem. Se poate observa că şi existenţa unei stări cu probabilitate 1 conduce la informaţie 0.

CUNOŞTINŢELE

Informaţiile constituie baza raţionamentelor, experimentărilor imaginate de mintea umană, în scopul obţinerii de noi *cunoştinţe*.

Cunoştinţele reprezintă esenţa ce rezidă în spatele datelor şi informaţilor într-un domeniu de activitate. Cunoştinţele se obţin prin cercetare ştiinţifică şi sunt un rezultat al sintezei procesării datelor, informaţiilor şi experienţei.

Cunoştinţele acumulate facilitează aprecierea relaţiilor dintre atributele unui element, obiect, sistem, proces. Relaţiile pot fi: procedurale, temporale, structurale, spaţiale, logice, semantice, etc. Pe baza înţelegerii relaţiilor se poate lua o decizie într-un context dat.

Capacitatea de a lua decizii corecte este influenţată de disponibilitatea datelor şi de aptitudinea de a anticipa consecinţele acţiunii. Capacitatea de a prezice într-un mod raţional consecinţele lucrărilor de intervenţie constituie cunoştinţa în domeniul managementului patrimoniului: cu cât este mai bogat corpul cunoştinţelor cu atât este mai puternic instrumentul de administrare; cu cât mai mare numărul de opţiuni cu atât decizia va avea o acurateţe mai ridicată.

Analiza mai multor variante şi managementul bazat pe cunoştinţe este alegerea logică şi totuşi prea puţine persoane sau compartimente de planificare efectuează analize explicite şi complete. Chiar şi atunci când se consideră mai multe opţiuni, judecata este superficială şi se opreşte după câţiva paşi. Procesul decizional este prezentat în figura următoare.

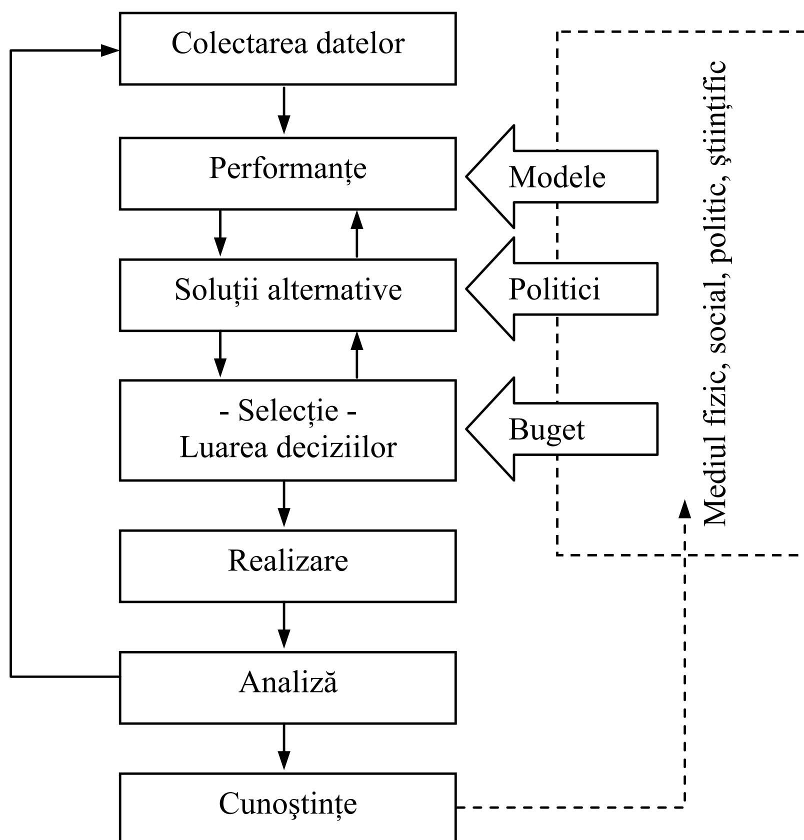


Fig. 2 Procesul decizional (după [26])

Există două cauze pentru care oamenii nu iau decizii bazate pe cunoştinţe: (1) deficienţă în cunoaştere şi (2) comportament inadecvat.

1. pentru a utiliza cunoştinţele este necesar ca acestea să existe, să fie disponibile şi să fie incluse în instrucţia personalului;
2. interese personale sau de grup, inerţia sau indiferenţa pot perturba procesul de decizie.

Pornind de la cele prezentate mai sus este necesară cercetare ştiinţifică pentru a furniza administratorului metode extinse de colectare şi interpretare a datelor, modele corespunzătoare şi algoritmi decizionali eficienţi. Opţiunile trebuie organizate, clasificate şi corect evaluate.

1.4. VALOAREA DATELOR, INFORMAŢIILOR ŞI CUNOŞTINŢELOR

Într-o lume cu magistrale informaţionale datele, informaţiile şi cunoştinţele au devenit mărfuri a căror comercializare cere sau se bazează pe metode de evaluare (Mowshowitz 1994 [20]).

În acest sens unii autori sunt extrem de categorici:

„Pentru a fi informație, informația trebuie să aibă o valoare, trebuie să poată fi utilizată în procesul decizional şi să fie proiectată să conducă la o acțiune... Ea trebuie să reducă incertitudinea celor care o obțin” (Machlup şi Mansfield 1983 [18], citat de Meadow şi Yuan, 1997 [19]).

Există mai multe rațiuni privind studierea valorii datelor, informațiilor şi cunoştințelor:

Motivația educativ-instructivă – studierea valorii poate conduce la identificarea acelor entități care trebuie luate în considerație în procesul de administrație fie prin utilizarea de către persoanele fizice implicate, fie prin includerea lor în sisteme automate, informatizate de luare a deciziei.

Motivația metodologică – metodele de evaluare a valorii informației şi cunoştințelor ajută la evaluarea sistemelor decizionale stabilite în urma cercetării sau experienței practice. Selecția între diferite tipuri de sisteme de inteligență artificială utilizate în rezolvarea problemelor se poate face prin măsurarea valorii cunoştințelor utilizate, generate şi manipulate.

Motivația organizatorică – se leagă de funcționarea sistemelor complexe în medii integrate, fie ele instituționale sau artificiale. Când mai multe module/compartimente sunt în competiție pentru rezolvarea unei sarcini, evaluarea cunoştințelor poate face identificare celui mai potrivit.

La nivelul actual, în administrarea infrastructurii transporturilor, deşi nu se recunoaşte, s-a făcut prea puțin pentru a stabili în mod ştiințific detaliile legate de colectarea datelor anume: cantitatea, precizia, frecvența şi chiar scopul. Cele mai importante rămân însă timpul şi costul şi aici probabil s-a făcut cel mai puțin. Pe de altă parte, industria transporturilor are o dinamică deosebită. Elementele de detaliu care trebuie măsurate şi prelucrate se schimbă destul de des. Colectarea datelor devine astfel o țintă în continuă mişcare.

Probabil că singurul domeniu care este reglementat din punctul de vedere al informației şi unde datele se culeg în continuu, după proceduri bine stabilite, este industria transporturilor aeriene, dar şi aici erorile, omisiunile, imprecizia afectează procesul, rezultatele fiind uneori catastrofale.

În domeniul administrării infrastructurii terestre s-au făcut paşi prea puțini şi prea mici ca importanță în vederea stabilirii raportului cost-beneficiu în ceea ce priveşte colectarea datelor. Fiind un proces pe termen lung, factorii de decizie care investesc în programe de colectare a datelor investesc probabil în ceva din care ei nu obțin nici un beneficiu. Majoritatea datelor rămân moştenire succesorilor. De aceea, de multe ori, factorii politici nici nu iau în calcul proiecte care presupun culegerea de date, urmărirea comportamentului în timp etc.

Pentru prelucrarea datelor şi extragerea de informaţii semnificative sunt necesare programe statistice performante. Acestea au în general preţuri mari, dar programele statistice de scară largă, care se pretează la prelucrarea datelor din infrastructura transporturilor, au costuri exorbitante. Fiind finanţate de la buget, majoritatea agenţiilor de administrare nu dispun de fondurile necesare pentru achiziţionarea şi menţinerea unor asemenea programe. În plus, justificarea necesităţii cheltuielilor pentru colectarea datelor ori însăşi colectarea datelor este deficitară. De multe ori nu se argumentează relevanţa datelor colectate şi implicaţiile cunoaşterii.

Conceptul de „relevanţă" şi măsurarea acesteia sunt deosebit de delicate. Tate-Glass ş.a. (2000 [30]) compară măsurarea relevanţei cu statul la focul de tabără: prea departe de „flacăra" politică rişti să fii tratat cu răceală, o poziţie prea apropiată prezintă riscul de a te arde din cauza cheltuielilor inutile.

Fundamental în rezolvarea problemei este anticiparea necesarului de date. Proiectele de statistică în infrastructura transporturilor, ca şi în alte domenii, sunt 5% statistică şi 95% logistică, ele fiind exerciţii complexe de organizare şi planificare. Elementul esenţial este în mai mică măsură competenţa în analize statistice cât abilitatea de a anticipa politicile şi nevoile de planificare în viitor.

Când problemele de planificare sau de strategie ajung la Ministerul Transporturilor spre studiu este deja prea târziu să începem colectarea de date. În planificări strategice trebuie să se lucreze cu acele date care sunt imediat disponibile.

Atunci când datele sunt cerute, conform Tate-Glass (2000 [30]), profesioniştii informaţiei pot răspunde în:

- trei minute, când datele sunt pe raft;
- trei ore, dacă trebuie căutat prin baza de date sau dosare;
- trei zile, când datele trebuie prelucrate;
- trei săptămâni, dacă trebuie făcută ceva programare sau modificat programul;
- trei luni, când este necesară prelucrarea completă a datelor;
- trei ani, dacă este necesară colectarea de noi date.

De aceea sunt necesare programe proactive de colectare a datelor care să vină în întâmpinarea viitoarelor analize de politici în transporturi.

1.5. UTILIZAREA DATELOR

Domeniul managementului infrastructurii rutiere este un domeniu intensiv în utilizarea datelor, informaţiilor şi cunoştinţelor. Date de teren, de structură, de materiale, de reţea, informaţii de evoluţie, despre origine destinaţie, cunoştinţe despre comportamentul agenţilor umani implicaţi în trafic, al vehiculelor şi echipamentelor. Iată doar câteva elemente care trebuie tratate de administratorii infrastructurii transporturilor.

Pentru a avea la dispoziţie date, informaţii şi cunoştinţe corecte şi complete în timp util administratorii infrastructurii transporturilor trebuie să aibă în vedere următoarele acţiuni:

- Identificarea necesităţilor de date;
- Crearea unui sistem de reprezentare a datelor;
- Implementarea şi evaluarea unui sistem de colectare a datelor;
- Crearea şi administrarea unui sistem de stocare şi regăsire a datelor
- Implementarea sistemului de transmitere a datelor;
- Identificarea algoritmilor utili în mecanismul decizional;
- Identificarea, implementarea şi evaluarea unor proceduri de prelucrare a datelor;
- Analiza rezultatelor în conformitate cu scopul urmărit
- Evaluarea sistemului de management a datelor.

Lucrarea de faţă urmăreşte şi tratează aceşti paşi, sau cel puţin încearcă să o facă. Totodată se încearcă să se prezinte laolaltă noţiunile şi metodele de procesare necesare pentru dezvoltarea unui sistem integrat de management a infrastructurii.

1.6. BIBLIOGRAFIE

[1] Bar-Hillel Y., Carnap R.: *An Outline of a Theory of Semantic Information*, 1953, rep. in Bar-Hillel 1964.
[2] Barwise J., Perry J.: *Situations and Attitudes*; MIT Press, Cambridge Mass. 1983.
[3] Braman S.: *Defining Information*; Telecommunications Policy 13, pp.233-242., 1989.
[4] Buckland M.: *Information as Thing*; Journal of the American Society of Information Science (42.5), pp.351-360, 1991.
[5] Checkland P. B., Scholes J.: *Soft Systems Methodology in Action*; John Wiley & Sons, New York 1990.
[6] Davis G. B., Olson M. H.: *Management Information Systems: Conceptual Foundations, Structure, and Development*; 2nd ed., McGraw-Hill, New York 1985.
[7] Devlin K.: *Logic and Information*; Cambridge University Press, Cambridge 1991.
[8] Dretske F.: *Knowledge and the Flow of Information*; MIT Press, Cambridge Mass. 1981, rep. Stanford: CSLI, 1999.
[9] Floridi Luciano: *Is Information Meaningful Data ?*; Electronic Conference on Foundations of Information Science (FIS 2002)
[10] Floridi Luciano: *Philosophy and Computing – An Introduction*; London – New York: Routledge, 1999.
[11] Grice P.: *Studies in the Way of Words*; Harvard University Press, Cambridge Mass. 1989.
[12] Howard R.A.: *Decision Analysis: Applied Decision Theory*; in Proceedings of the Fourth International Conference on Operational Research, D. B. Hertz and J. Melese, eds., Wiley, New York, 1966, pp. 55-71.
[13] Ionescu Constantin, Scînteie Rodian: *Informational Flows in Bridge Engineering*; in Ioani A., Chira C., Manea D. (editors) Proceedings of the International Conference, Constructions 2003, Volume 4 – Road, Bridges and Railways, pp. 209-216, Argonaut&Napoca Star, Cluj-Napoca, Romania, May 2003.

[14] Israel D., Perry J.: *What is Information?*; in P. Hanson (ed.), Information, Language and Cognition, pp.1-19, University of British Columbia Press, Vancouver 1990.
[15] Losee R. M: *A Discipline Independent Definition of Information*; Journal of the American Society for Information Science, 48.3, 254-269, 1997.
[16] Lucey T.: *Management Information Systems*; 6th ed. DP Publications Ltd, London 1991.
[17] Machlup F.: *Semantic Quirks in Studies of Information*; in Machlup, F. And Mansfield, U. eds.. The Study of Information: Interdisciplinary Messages, pp. 641-671 John Wiley, New York, 1983.
[18] Machlup Fritz, Mansfield Una: *The Study of Information: Interdisciplinary Messages*; John Wiley, New York, 1983.
[19] Meadow C. T., Yuan, W.: *Measuring the impact of information: defining the concepts*; Information processing and management. Vol. 33, no. 6, pp. 697-714, 1997.
[20] Mowshowitz A.: *Information as a commodity: assessment of market value*; In Yovits, M. C., editor, Advances in Computers, Vol. 38, pages 247–316, London. Academic Press., 1994.
[21] NATO 1974, Advanced Study Institute in Information Science, Champion, 1972. *Information Science: Search for Identity*; ed. by A. Debons (New York: Marcel Dekker).
[22] NATO 1975, Advanced Study Institute in Information Science, Aberystwyth, 1974. *Perspectives in Information Science*, ed. by A. Debons and W. J. Cameron. (Leiden: Noordhoff).
[23] NATO 1983, Advanced Study Institute in Information Science, Crete, 1978. *Information Science in Action: Systems Design*, ed. by A. Debons and A. G. Larson. (Boston: Martinus Nijhoff).
[24] Păunescu, F., Badea-Dincă, N., Stăcuţ., E., *Informatizarea societăţii: un fenomen inevitabil?*, Ed. Şiinţifică şi Enciclopedică, Bucureşti, 1985.
[25] Schrader, A.: *In Search of a Name: Information Science and its Conceptual Antecedents*; Library and Information Science Research 6, pp.227-271, 1984.
[26] Scînteie Rodian: *Decision Making Process in Highways Structures Management*; in Ioani A., Chira C., Manea D. (editors) Proceedings of the International Conference, Constructions 2003, Volume 4 – Road, Bridges and Railways, pp. 287-294, Argonaut&Napoca Star, Cluj-Napoca, Romania, May 2003.
[27] Shannon, C. E.: *Collected Papers*; ed. N. J. A. Sloane & A. D. Wyner, Los Alamos, CA, IEEE Computer Society Press, 1993.
[28] Silver G. A., Silver M. L.: *Systems Analysis and Design*; Addison Wesley, Reading MA, 1989.
[29] Szaniawski, K.: *On Defining Information*; 1984, rep. in Szaniawski 1998.
[30] Tate-Glass Martha J., Bostum Rob, Witt Greg: *Data, Data, Data—Where's the Data?*; TRB – Millennium Papers, National Research Council, Transportation Research Board, Washington, D.C., USA, January 2000.
[31] Warner T.: *Communication Skills for Information Systems*; Pitman, London 1996.
[32] Wellisch, H., *From Information Science to Informatics*; Journal of Librarianship 4, pp.157-187, 1972.
[33] Wersig G., Neveling U.: *The Phenomena of Interest to Information Science*; Information Scientist 9, 127-140, 1975.
[34] Zadeh LA: *Toward a Theory of Fuzzy Information Granulation and its Centrality in Human Reasoning and Fuzzy Logic*; Fuzzy Sets and Systems, vol. 90, No.2, pp.111-127, 1997.

Capitolul 2

STRUCTURI DE DATE

Rezolvarea problemelor cu ajutorul calculatorului presupune analiza, gruparea şi structurarea datelor pentru a obţine eficienţă maximă. Utilizarea structurilor de date este impusă pe de o parte de înţelegerea facilă a problemei şi pe de altă parte de necesitatea implementării practice a procedurilor de rezolvare a problemei pe maşinile fizice.

Structurile de date reprezintă o cale sistematică de organizare şi accesare a datelor [36].

Reprezentarea datelor şi a structurilor de date depind în mare măsură de construcţia calculatoarelor fizice şi de implementările limbajelor de programare. Totuşi pe majoritatea calculatoarelor şi în majoritatea limbajelor apar următoarele structuri de date: setul, articole (înregistrarea), tablouri, liste, urne (buckets), stive, cozi, arbori şi reţele.

În prezenta lucrare suntem preocupaţi mai puţin de reprezentarea efectivă a datelor în memorie sau de tipurile implicite oferite de limbajul de programare. De aceea prima structură tratată este articolul sau înregistrarea. Aceasta este baza pentru toate celelalte reprezentări de tipuri de date abstracte.

2.1. ARTICOLUL

Articolul sau înregistrarea este o structură neomogenă reprezentând reuniunea unui număr fix de componente numite câmpuri care constituie logic o unitate. Câmpurile din structură sunt caracterizate prin tipul lor care poate fi unul primar, oferit de limbajul de programare, sau unul complex, definit de către proiectant (deci la rândul său un articol). Gruparea câmpurilor se face din considerente de procesare dar

în cele mai multe cazuri sunt grupate atribute comune ale unei entităţi care trebuie tratată. Lungimea totală a articolului este dată de suma lungimilor câmpurilor ($L(A) = \sum_i L_i$) (vezi Fig. 3).

Articol A

Câmp_1	Tip_câmp_1	L1
Câmp_2	Tip_câmp_2	L2
Câmp_3	Tip_câmp_3	L3
:	:	
Câmp_i	Tip_câmp_i	Li
:	:	
Câmp_n	Tip_câmp_n	Ln

Fig. 3 Structura unui articol

Regăsirea câmpului se face cu ajutorul unui identificator asociat la definire. Pentru regăsirea unui câmp când există mai multe articole se utilizează numele articolului şi al câmpului împreună.

Implementările moderne ale obiectelor pot genera articole cu structură variabilă. Chiar dacă unele dintre variante au lungimi diferite, articolul în ansamblu va avea lungimea celei mai lungi dintre variante şi aceasta este cantitatea de memorie care va fi rezervată de fiecare dată când se creează un nou articol.

2.2. TIPURI ABSTRACTE DE DATE

O categorie specială de structură de date derivată din articol este **obiectul**. Un obiect, denumit în unele implementări recente **clasă**, încapsulează într-o structură unică atât datele cât şi un set de proceduri şi funcţii care efectuează prelucrarea de bază a datelor respective. Astfel de proceduri, ataşate în momentul definirii, elimină ambiguitatea şi oferă siguranţa că prelucrarea şi rezultatele obţinute se referă strict la setul de date conţinut de obiect.

Procedurile definite în interiorul obiectului ajută proiectantul să nu se mai gândească la procesul complicat de referire a câmpurilor şi regăsire a datelor care apare când acestea sunt căutate din exterior. Modalităţile efective de definire a obiectelor şi a procedurilor şi funcţiilor ataşate depind de limbajul de programare ales. Definirea unor noi obiecte se poate face pe baza unora deja existente. Noile obiecte moştenesc structura de date şi proprietăţile de la cele anterioare. Ele pot defini noi elemente sau le pot modifica pe cele anterioare.

Încapsularea componentelor şi moştenirea permit un anumit grad de abstractizare a datelor (*a abstractiza cu sensul de a distila un sistem complicat până*

la conceptele sale fundamentale). Aceasta implică identificarea componentelor şi descrierea funcţionalităţii lor. Datele sunt organizate modular şi există posibilitatea ca detaliile interne de implementare să fie „ascunse" utilizatorilor, iar comunicarea să se facă numai prin acea parte a datelor şi funcţiilor care sunt declarate publice. Se obţine astfel un grad ridicat de libertate pentru analişti şi programatori.

Aplicând paradigma abstractizării la proiectarea structurilor de date obţinem tipuri abstracte de date - TAD.

Tipul abstract de date este un model matematic prin care se definesc tipurile de date ce pot fi stocate într-o structură şi metodele disponibile să proceseze datele. Se defineşte deci o interfaţă. Interfaţa este un „contract" prin care ni se garantează disponibilitatea unor elemente, fără să ni se ofere detalii privind implementarea internă [36].

Tip Abstract de Date = Interfaţă + Implementare (2)

Interfaţa defineşte proprietăţile logice ale TAD şi în special profilul sau semnătura operaţiilor sale.

Implementarea defineşte reprezentarea structurii de date şi algoritmii de implementare a operaţiilor.

Un TAD este realizat printr-un pachet, cel mai adesea un pachet generic. Particularizările se fac prin definirea de descendenţi care preiau (moştenesc) caracteristicile tipului iniţial.

Specificaţiile pachetului sunt reprezentate de interfaţă. Structura de date este declarată ca o zonă privată. Operaţiile sunt declarate sub forma unor subprograme ce au cel puţin un parametru.

Partea privată a specificaţiilor şi a corpului pachetului reprezintă implementarea TAD. Tot aici este inclusă şi reprezentarea structurii de date.

Exemple de tipuri de operatori [48]:

- Constructori – crearea şi iniţializarea unui obiect.
- Selectori – furnizează informaţii despre starea obiectului.
- Modificatori – alterează starea unui obiect.
- Destructor – distruge obiectul.
- Iteratori (engl. iterators; franc. parcoureurs, itérateurs) – accesează toate părţile unui obiect compus şi aplică o anumită acţiune fiecăruia dintre ele.

Studiu de caz

Un exemplu de limbaj de programare ce permite declararea şi lucrul cu tipuri abstracte de date este şi dialectul de Pascal al mediului de programare Delphi dezvoltat de Borland.

```
type
  TNewObject = class(TOldObject)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  end;
```

Fig. 4 Declararea claselor în Delphi Pascal

Un membru (element) **privat** (engl. *private*) este invizibil în afara unităţii de programare sau modulului în care clasa a fost declarată. Prin plasarea declaraţiilor claselor legate între ele în acelaşi modul puteţi să oferiţi claselor acces una la resursele alteia fără ca elementele respective să fie „văzute" din afară de alte clase sau proceduri.

Un membru **protejat** (engl. *protected*) este vizibil în modulul în care clasa a fost declarată şi în oricare clasă descendentă, indiferent de modulul în care aceasta apare. Se intenţionează ca aceste elemente să fie caracteristici generale pentru toţi descendenţii.

Un membru **public** este vizibil oriunde clasa este vizibilă şi poate fi referită.

Un membru **publicat** (engl. *published*) are aceeaşi vizibilitate ca orice membru public. Diferenţa constă în faptul că pentru aceste elemente se generează informaţii suplimentare pe timpul rulării (RTTI – *runtime type information*). Prin aceasta se pot asocia metode specifice cu proprietăţi specifice (de ex. proceduri de tratarea evenimentelor cu evenimente particulare).

2.3. SETUL

Setul sau mulţimea reprezintă o colecţie finită de elemente $M = \{m_1, m_2, \ldots, m_n\}$. Deoarece, prin definiţie, este finit, setul poate fi imaginat ca un vector. Deoarece componenţa sa este aprioric definită şi cunoscută, reprezentarea ca vector de tipul $(m_1, m_2, \ldots, m_n)$ nu ar aduce informaţii relevante. Este mult mai important să reprezentăm un sub-set *S* ce poate fi definit pe mulţimea iniţială.

Pentru aceasta vom defini o funcţie de apartenenţă $A : M \to \{0,1\}$, astfel:

$$a_i = A(m_i) = \begin{cases} 1, \text{când } m_i \in S \\ 0, \text{când } m_i \notin S \end{cases} \quad (3)$$

Acum sub-setul poate fi reprezentat prin vectorul valorilor funcţiei de apartenenţă $v_S = (a_1, a_2, \ldots, a_n)$.

Operațiile ce trebuie să fie create pe un sub-set sunt:

- **Inserare**, pentru a introduce un element în sub-set [Insert(e,S) $\Rightarrow$e$\in$S$\notin$];
- **Eliminare**, care scoate elementul din sub-set [Suppress(e,S) $\Rightarrow$e$\notin$S];
- **Aparține**, funcție care face cunoscut apartenența unui element la sub-set;
- **Vid**, indică faptul că sub-setul nu conține nici-un element;
- Alte operații ce se pot defini sunt:
  - complementul,
  - reuniunea,
  - intersecția,
  - diferența,
  - diferența simetrică.

2.4. TABLOUL

Tabloul reprezintă, în esență, o mulțime de date de acelaşi tip care se grupează în poziții succesive în aşa fel încât să poată fi regăsite prin utilizarea unor indici de poziție.

Astfel aplicația:

$$f:\{1,2,\cdots,n_1\}\times\{1,2,\cdots,n_2\}\times\cdots\times\{1,2,\cdots,n_k\}\rightarrow M \quad (4)$$

care ataşează fiecărui k-uplu de indici $(i_1,i_2,\cdots,i_k)$ un element $m_{i_1,i_2,\cdots,i_k}\in M$ defineşte un tablou *k*-dimensional. Pentru cazul particular $k=1$ se obține un vector. Reprezentarea fizică a tablourilor se face prin rezervarea de locații succesive într-o zonă contiguă de memorie.

Identificarea unui anumit element din tablou se face prin numele tabloului şi setul de indici corespunzători elementului. Aşa cum spuneam la început tabloul se regăseşte ca o structură omogenă şi oricare dintre operațiile definite se aplică oricărui element din tablou. Pe astfel de structuri se pot lesne aplica proceduri repetitive. Implementările moderne ale tablourilor permit definirea unor seturi de indici care nu pornesc obligatoriu de la 1 sau 0, lucru ce dă mai multă flexibilitate în programare.

Elementele din tablou pot fi de tip simplu oferite de către limbajul de programare (întregi, reali, şiruri de caractere etc.) dar şi tipuri complexe definite de către utilizator (ex. articole, obiecte etc.).

În principiu, lungimea unui tablou este fixă şi deci numărul de elemente este constant. În general, pentru a simula un număr variabil de elemente se contorizează într-o variabilă externă numărul de elemente ocupate sau se utilizează tablouri de pointeri pentru care se alocă ulterior articole sau obiecte.

2.5. LISTA ÎNLĂNȚUITĂ

Lista este o structură de date care are lungime variabilă. În multe cazuri, nu se poate cunoaşte din momentul proiectării programului care va fi numărul exact al elementelor utilizate în calcul.

Alocarea unui anumit spațiu fixat în faza de programare poate conduce la două situații neplăcute: fie se alocă spațiu insuficient şi problema rămâne nerezolvată, fie se alocă prea mult spațiu care rămâne neutilizat şi se consumă inutil din resursele calculatorului blocând celelalte programe. Soluția propusă a fost de a aloca dinamic memorie pe măsură ce nevoia o impune. În structura articolelor, pe lângă informația utilă, se păstrează referințe către alte articole similare care pot fi create pe măsură ce programul este rulat. Se creează astfel o înlănțuire care are o lungime variabilă, programul utilizând atâta memorie cât este necesară.

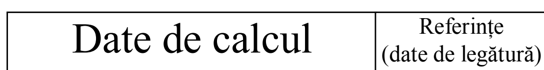


Fig. 5 Structura unui obiect dintr-o listă

Înlănțuirea de articole similare care conțin referințe unul despre altul păstrând un singur nivel de legături se numeşte listă (Fig. 6).

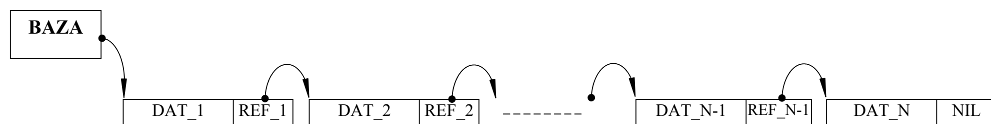


Fig. 6 Înlănțuirea în listă

Lista este accesată din interiorul programului prin intermediul unei adrese de bază (o referință, un pointer etc.), elementul din listă care este referit fiind capul de listă. Se observă că datele referință din ultimul element din listă sunt vide (NIL), acesta fiind unul dintre indicatorii de terminare a înlănțuirii.

Din punct de vedere matematic, lista reprezintă o secvență de zero sau mai multe elemente de un anumit tip dat. Numărul n de elemente ce se regăsesc în listă reprezintă **lungimea listei**. Când lungimea este 0 adresa de bază este vidă (NIL).

Elementele pot fi identificate prin poziția lor în listă. Din considerente ce țin de reprezentarea numerelor întregi în formatul binar este uzual a se spune că primul element ocupă poziția 0.

Dacă ultimul element conține o referință către capul de listă avem o listă circulară. Înlănțuirea poate fi simplă, dar se poate ca fiecare element să conțină referințe atât către succesor cât şi către predecesor. În acest caz înlănțuirea este dublă.

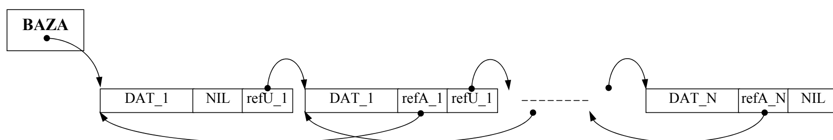


Fig. 7 Listă cu înlănţuire dublă

Posibilitatea creării de liste şi, în general, proprietăţile pe care au fost identificate pentru articole se păstrează şi pentru obiecte.

Pe liste se pot defini unele operaţiuni:

- Adaugă(x,L) / Add(x,L). Adaugă elementul x la lista L. Poziţia va fi la coada listei. Lungimea listei se incrementează.
- Introduce(x,p,L) / Insert(x,p,L). Include elementul x în cadrul listei L în poziţia p.
- Caută(x,L) / Find(x,L). Returnează poziţia elementului x în lista L. Dacă x nu este inclus în listă atunci rezultatul este sfârşitul listei. Dacă x este inclus mai mult decât o singură dată atunci prima apariţie întâlnită este întoarsă ca rezultat.
- Element(p,L) / Retrive(p,L). Returnează elementul aflat pe poziţia p în lista L. Rezultatul este nedefinit (Nul) dacă p este mai mare sau egal cu lungimea listei. Tipul rezultatului este tipul elementului este acelaşi cu tipul definit pentru elementele listei.
- Elimină(p,L) / Delete(p,L). Şterge elementul din poziţia p din lista L. Operaţia afectează doar lista, elementul rămâne în memorie şi poate fi accesat dacă este referit prin alte metode (de ex. mai este inclus într-o altă listă). Operaţia trebuie utilizată cu atenţie pentru a nu lăsa obiecte fără cale de acces.
- Prim(L) / First(L). Întoarce poziţia primului element din lista L. Permite utilizarea operaţiei ***Următor*** (engl. Next).
- Ultim(L) / Last(L). Întoarce poziţia ultimului element din listă.
- Urmator(L) / Next(L). Întoarce poziţia elementului din lista L care urmează după elementul selectat. Dacă în listă există mai puţin de două elemente sau elementul selectat este ultimul din listă atunci rezultatul este nedefinit.
- Anterior(L) / Preview(L). Întoarce poziţia elementului din lista L care se situează înaintea elementului selectat. Dacă în listă există mai puţin de două elemente sau elementul selectat este primul din listă atunci rezultatul este nedefinit.
- Sterge(L) / Clear(L). Asemănător cu eliminarea dar pe lângă ştergerea referirii din listă se încheie şi existenţa obiectului. După această operaţie obiectul nu mai poate fi accesat. Atenţie: dacă obiectul mai era referit prin alte mecanisme (de ex. era inclus în altă listă) apelarea va genera erori fatale.

Mai jos prezentăm câteva exemple de implementări de operaţii pe liste.

```
Proc Inser( x : DataType, L: ListType )
    x.Next <- head
    If head<>Nil then Head.Prev <- x
    Head <- x
    x.preview <- Nil
EndProc
```

Fig. 8 Implementarea funcţiei de inserarea

În Fig. 6 s-a prezentat o operaţie de inserare. S-a presupus implicit că inserarea se face în capul listei (prima poziţie).

Corespunzător operaţiei de adăugare, s-a creat şi operaţia de eliminare. În figura următoare prezentăm un exemplu de implementare.

```
Proc Delete( x : DataType, L: ListType )

    If x.preview ≠ Nil
       Then x.preview.next ← x.next
       Else L.Head ← x.next
    EndIf
    If x.Next ≠ Nil
       Then x.Next.Preview ← x.Preview
    EndIf

EndProc
```

Fig. 9 Ştergerea unui element dintr-o listă înlănţuită

2.6. URNE

Urnele sau grămezile (literatura de limbă engleză utilizează termenul de *bucket*) reprezintă o grupare de date a căror ordine de utilizare nu prezintă importanţă.

2.7. TABELE DE DISPERSIE (HASH TABLES)

Tabelele de dispersie (hash tables) reprezintă o metodă eficientă de implementare a dicţionarelor. Pentru căutare ele asigură o complexitate $O(1)$ în medie şi $O(n)$ în cazul cel mai defavorabil [37], [43].

În practică, tabelele de dispersie sunt simpli vectori adresaţi de o funcţie de repartiţie. Fiecare celulă a vectorului conţine o referinţă (pointer) către o listă înlănţuită în care se plasează elementele dicţionarului.

Prezentăm în continuare un exemplu de tabelă de dispersie cu dimensiunea 5.

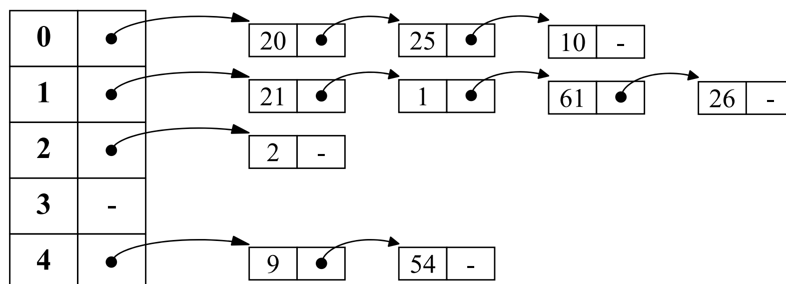


Fig. 10 Tabelă de dispersie

Cazul cel mai defavorabil este dat de situarea tuturor elementelor într-o singură listă înlănţuită.

Un asemenea caz este dat sau evitat de forma funcţiei de repartiţie. Dacă funcţia de repartiţie este uniformă atunci lista iniţială se subdivide în părţi aproximativ egale. Pentru aceasta se utilizează diverse tipuri de funcţii:

- Restul împărţirii la M (operaţia modulo M). Cheia considerată pentru fiecare element se împarte la dimensiunea tabelei. Restul rezultat se utilizează ca index pentru poziţionarea în tabelă.
- Se alege M să fie număr prim. Se asigură o uniformitate mai bună.

O tabelă de dispersie, ca structură de stocare a datelor are un constructor şi trei metode:

- Creare / Init creează tabela şi iniţializează câmpurile la valoarea implicită;
- Inserează(M) / Insert(M) calculează indexul $i = index(M)$ cu ajutorul funcţiei de dispersie şi introduce elementul în lista adresată de poziţia i din tabelă;
- Caută(M) / Find(M) calculează indexul $i = index(M)$ cu ajutorul funcţiei de dispersie şi caută elementul în lista adresată de poziţia i din tabelă; dacă nu este găsit se returnează Nil sau se generează o eroare;
- Elimină(M) / Remove(M) calculează indexul $i = index(M)$ cu ajutorul funcţiei de dispersie, caută elementul în lista adresată de poziţia i din tabelă şi elimină obiectul; dacă nu este găsit se returnează Nil sau se generează o eroare;

2.8. STIVE

Stivele reprezintă structuri de date care au o singură cale de acces atât pentru introducere cât şi pentru extragere de date: vârful stivei. Pentru ele se alocă iniţial o

zonă statică, fixă a cărei conţinut este vid (numărul de elemente este 0). Pe măsură ce noi date trebuie păstrate ele se depun în stivă numărul elementelor fiind incrementat (Fig. 11). Când se doreşte utilizarea datelor salvate singura modalitate de acces permisă este scoaterea elementului din stivă (cu decrementarea numărului de elemente) (Fig. 12).

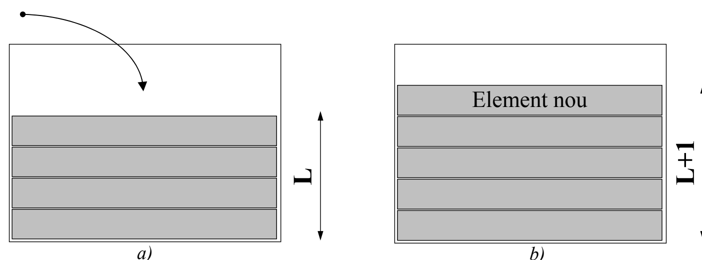


Fig. 11 Stiva înainte şi după introducerea unui element

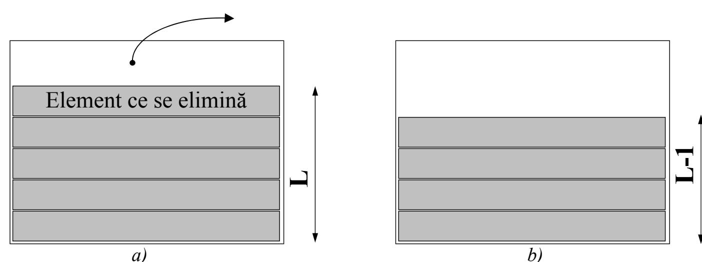


Fig. 12 Stiva înainte şi după extragerea unui element

Modul de lucru al acestor structuri este *ultimul intrat – primul ieşit* LIFO (last in - first out). Pentru o corectă funcţionare, în general, se alocă un tablou cu elemente de tipul cerut şi o variabilă în care se păstrează un indicator de stivă. Valoarea 0 a indicatorului semnalizează stiva goală. Valoarea indicatorului egală cu lungimea stivei semnalizează stiva plină. Verificarea acestor două stări cade în sarcina programatorului înainte de a efectua extragerea şi respectiv depunerea.

La adăugarea unui nou element valoarea indicatorului creşte cu unu ($L \leftarrow L+1$) în timp ce la scoaterea unui element valoarea indicatorului de stivă descreşte cu unu ($L \leftarrow L-1$).

Pe o asemenea structură se pot defini următoarele operaţii minimale:

- Sterge(S) / Clear(S): elimină toate datele din stivă şi aduce indicatorul în poziţia iniţială.
- Adauga(x,S) / Push(S): introduce elementul x în vârful stivei. Indicatorul este incrementat. Dacă stiva este deja plină generează o eroare.

- Extrage(S) / Pop(S): extrage elementul din vârful stivei şi îl întoarce ca rezultat. Indicatorul este decrementat corespunzător. Dacă stiva este goală generează un mesaj şi se returnează elementul vid.
- Varful(S) / Top(S): verifică elementul din vârful stivei fără a-l îndepărta.
- EsteVida(S) / IsEmpty(S): verifică dacă stiva este goală şi întoarce valoarea logică (booleană) corespunzătoare.

O implementare facilă a stivei pentru tipuri complexe de articole se poate realiza prin utilizarea unei liste înlănțuite. Descrierea în limbaj pseudocod se face în figura următoare (Fig. 13).

```
DecType DataType As Record
          ......
        EndRecord
DecType LinkedRecord As Record
              Data : DataType
              Next : ^LinkedRecord
        EndRecord
Declare Heap As LinkedRecord
Proc Push( x : DataType )
    tmp <- new LinkedRecord
    tmp.data <- x
    tmp.next <- Heap
    Heap <- tmp
EndProc
Function Pop():
    If IsEmpty() then Error
    tmp <- top
    top <- top.next
    tmpX <- tmp.data
    Delete tmp
    Return tmpX
EndFunct
Funct IsEmpty() :
    if Heap = Nil
        then Return TRUE
        else Return FALSE
EndFunct
```

Fig. 13 Operațiile pentru stiva implementată prin listă înlănțuită

2.9. COZI

Coada reprezintă o structură de date în care accesul se realizează pe două căi: una de intrare şi una de ieşire. Atunci când coada nu este vidă, elementele sunt disponibile la ieşire în ordinea în care au fost depuse. Modul de lucru al cozilor este

primul intrat – primul ieşit FIFO (first in - first out). Reprezentarea intuitivă a modului de lucru este prezentată în Fig. 14 şi Fig. 15.

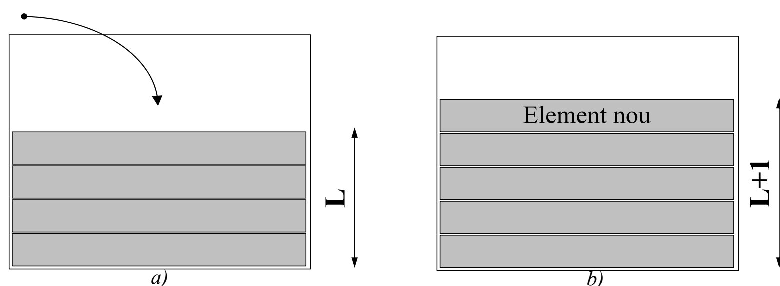


Fig. 14 Coada înainte şi după introducerea unui element

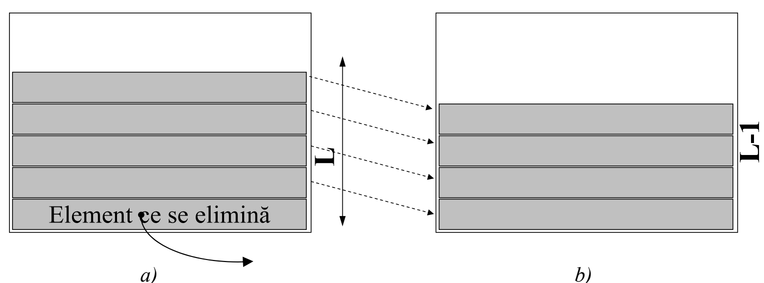


Fig. 15 Coada înainte şi după extragerea unui element

În fapt, pentru implementarea unei cozi se poate utiliza tot un tablou care se alocă cu o lungime fixă de *n* elemente şi două variabile care păstrează doi indicatori: unul privind sfârşitul cozii (*Ia*) şi unul pentru vârful cozii (*Ie*) (vezi Fig. 16).

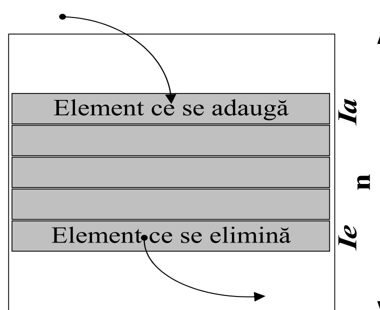


Fig. 16 Implementarea practică a cozilor

Presupunând *Ia* şi *Ie* indici în tablou, utilizăm *Ia* pentru adăugare şi *Ie* pentru extragere. După adăugarea unui nou element se incrementează *Ia* ($Ia \leftarrow Ia+1$).

Dacă *Ia* depăşeşte dimensiunea tabloului *Ia* ia valoarea bazei tabloului (pentru simplitate să presupunem 0). Când se extrage un element se incrementează *Ie* ($Ie \leftarrow Ie + 1$) şi, în mod similar, dacă *Ie* depăşeşte dimensiunea tabloului atunci ia valoare bazei tabloului. Dacă $Ia = Ie$ atunci coada este vidă, dacă $Ie \leftarrow (Ia+1)(\bmod n)$ atunci coada este plină. Este datoria programatorului să facă verificarea înainte de fiecare extragere şi adăugare respectiv.

Pentru cozi se pot defini următoarele operații minimale:

- Sterge(Q) / Clear(Q): elimină toate datele din coadă şi aduce indicatorii Ia şi Ie în poziția inițială.
- Adauga(x,Q) / Enqueue(x,Q): Introduce elementul x la sfârşitul cozii. Indicatorul Ia este incrementat. Când coada este deja plină generează o eroare fără să mai adauge elementul.
- Extrage(Q) / Dequeue(Q): extrage elementul din vârful cozii şi îl întoarce ca rezultat. Indicatorul Ie este incrementat corespunzător. Dacă coada este goală generează un mesaj şi se returnează elementul vid.
- Varful(Q) / Top(Q): verifică următorul element din coadă fără a-l îndepărta.
- EsteVida(Q) / IsEmpty(Q): verifică dacă coada este goală şi întoarce valoarea logică (booleană) corespunzătoare.
- EstePlina(Q) / IsFull(Q): verifică dacă coada este goală şi întoarce valoarea logică (booleană) corespunzătoare.

Operațiile de mai sus pot fi simplu descrise utilizând limbajul pseudocod. Prezentăm mai jos câteva dintre funcții pentru implementarea cu ajutorul tabloului.

```
Declare Q As array of OType
Declare Ia, Ie As Integer
Proc Enqueue(x)
    Ia <-(Ia+1) mod N
    Q[Ia] <- x
EndProc
Funct Dequeue () :
    Ie <-(Ie+1) mod N
    Return Q[Ie]
EndFunct
Funct IsEmpty() :
     If Ie = Ia
        then Return TRUE
        else Return FALSE
EndFunct
Funct IsFull() :
     If (Ia+1) mod N = Ie
        then Return TRUE
        else Return FALSE
EndFunct
```

Fig. 17 Operațiile pentru o coadă implementată printr-un tablou

Reprezentarea cozii prin liste înlănţuite:

```
DecType DataType As Record
          ......
        EndRecord
DecType LinkedRecord As Record
              Data : DataType
              Next : ^LinkedRecord
        EndRecord
DecType Queue As Record
              Head, Tail: ^LinkedRecord
        EndRecord
Declare Q As ^Queue
Funct IsEmpty( Q ) :
   If Q = Nil
      Then return True
      Else Return False
EndFunct
Proc enqueue(Q, x)
    tmp ← new LinkedRecord
    tmp.data ← x
    tmp.next ← Nil
    If Q.Head = Nil
       Then Q.Head ← tmp
       Else (Q.Tail).Next ← p
    Q.Tail ← p
EndProc
Funct dequeue( Q )
    If IsEmpty(Q) then Error

    Tmp ← Q.Head
    TmpX ← Tmp.Data
    Q.Head ← Tmp.Next
    If Q.Head = Nil Then Q.Tail ← Nil
    Free Tmp
    Return TmpX
EndFunct
```

Fig. 18 Reprezentarea cozilor prin liste înlănţuite

2.10. ARBORI

Considerăm o mulţime finită de elemente (fie ele articole sau obiecte) pe care le vom denumi noduri. Această mulţime se va numi ***arbore*** dacă :

- există un nod special denumit rădăcină;
- celelalte noduri sunt grupate în submulţimi care la rândul lor sunt arbori.

Elementele arborelui sunt nodurile şi legăturile care se stabilesc între acestea.

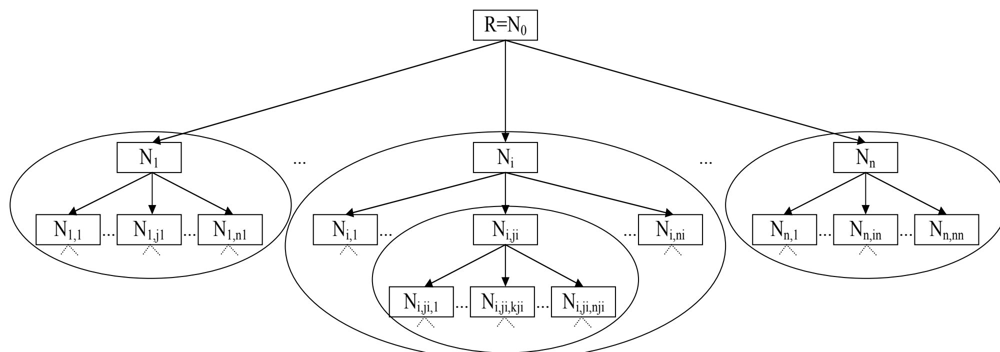


Fig. 19 Arbore

Rădăcina constituie nivelul 0 şi are referiri către un număr de noduri care constituie nivelul 1, fiecare dintre nodurile acestui nivel au referiri către nodurile unui al doilea şi aşa mai departe. Fiecare nod este direct subordonat doar unui singur alt nod. Orice nod este ***părinte*** pentru subordonaţii săi care, la rândul lor sunt ***descendenţi*** pentru acesta. Orice nod fără descendenţi este ***nod terminal*** sau ***frunză***.

Înălţimea (sau adâncimea) unui arbore este dată de numărul de niveluri ierarhice prezente.

Structura de date trebuie deci să conţină pe lângă datele utile şi referinţe către toţi descendenţii şi eventual către părinte.

Date de calcul	*Părinte*	*REF 1*	*REF 2*	...	*REF n*

Fig. 20 Structura unui nod dintr-un arbore

Mai sus prezentăm schematic structura unui nod al unui arbore. Referinţele către părinte nu sunt obligatorii. Nu se fac referiri către noduri cuprinse în alţi sub-arbori ai aceluiaşi arbore. Dacă există descendenţi sau nu există suficienţi care să completeze spaţiul lăsat pentru referinţe, locaţiile excedentare se completează cu NIL (entitatea care identifică lista vidă). Dacă referinţa PĂRINTE există şi este egală cu NIL avem de-a face cu rădăcina.

ARBORI BINARI

Reprezintă o categorie specială de arbori. Fiecare dintre noduri are doar maxim doi descendenţi: un sub-arbore stânga şi un sub-arbore dreapta. Poziţia celor doi este foarte importantă spre deosebire de arborii normali la care ordinea fiilor nu contează şi era suficient să se cunoască cine este descendentul cui.

Pentru arborii binari trebuie cunoscută atât descendenţa cât şi poziţia corectă. Cei doi sub-arbori nu sunt interschimbabili.

Structura de date a unui arbore binar este prezentă în figura următoare:

Date de calcul	*Părinte*	*Stânga*	*Dreapta*

Fig. 21 Structura unui nod dintr-un arbore binar

Dacă unul sau ambii descendenţi lipsesc locaţia se completează cu NIL.

Arborii binari sunt foarte populari în implementarea mecanismelor de căutare a datelor ordonate.

Câteva definiţii se impun în acest context:

Arbore strict binar – un arbore binar în care fiecare nod care nu este frunză are ambii descendenţi;

Arbore binar complet – arbore în care toate frunzele sunt la acelaşi nivel;

Arbore binar de căutare – arbore binar care nodurile sunt ordonate pentru a facilita operaţia de căutare.

Arbori binari de căutare

Este un tip special de arbore binar în care datele sunt aranjate cu respectarea unor reguli. Presupunând că valoarea unui nod oarecare dintr-un arbore binar de căutare este *m*, următoarele proprietăţi sunt întâlnite [37]:

- Toate valorile mai mici decât *m* sunt în stânga,
- Toate valorile mai mari decât *m* sunt în dreapta.

Spre exemplu, dacă avem 7 numere ele vor fi aranjate într-un arbore binar de căutare astfel:

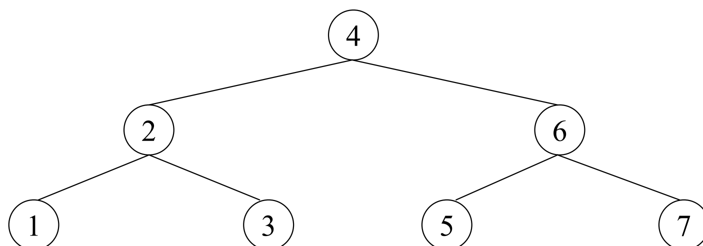


Fig. 22 Reprezentarea datelor într-un arbore binar de căutare

Înălţimea arborelui din exemplul precedent este 3. se poate observa faptul că dacă vom căuta o valoare în arbore vom găsi după maxim 3 paşi.

Se poate observa că valorile sunt aşezate în mod regulat în aşa fel încât să se obţină un echilibru. Aceste echilibru menţine o eficienţă maximă în utilizarea

arborelui de căutare. Spre deosebire de exemplul de mai sus în figura următoare avem reprezentat un arbore dezechilibrat.

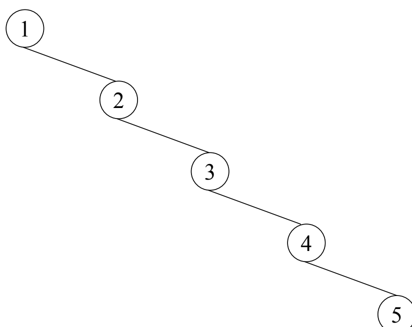


Fig. 23 Reprezentarea datelor într-un arbore dezechilibrat

Acest arbore este binar de căutare, el respectând toate regulile impuse. Din nefericire toate valorile au fost depuse în descendenții de pe un singur sens. Pentru doar cinci valori s-a obținut o înălțime a arborelui de cinci ceea ce face căutarea ineficientă.

Din aceste considerente s-a impus utilizarea altor tipuri de arbori.

Arbori roşu-negru

Sunt arbori binari de căutare în care se doreşte o balansare cât mai bună a datelor. Numele vine de la faptul că nodurile se colorează cu două culori: roşu şi negru. Aceste culori sunt atribuite după reguli care determină echilibrarea arborelui.

Arborii roşu-negru au următoarele proprietăți [37]:

- Fiecare nod este colorat sau în negru sau în roşu;
- Orice frunză este un nod NIL şi se colorează în negru;
- Dacă un nod este roşu atunci ambii descendenți sunt negri;
- Orice cale simplă de la un nod la o frunză descendentă conține acelaşi număr de noduri negre.

Numărul de noduri colorate în negru de pe o cale poartă numele de „adâncime neagră” sau „înălțime neagră”.

ARBORI B

Arborii B (Knuth 1998 [43], Cormen 1998 [37], Aho 1983 [35]) reprezintă o altă cale de implementare a unui mecanism de menținere echilibrată a arborilor de căutare. El reprezintă o garanție a faptului că arborele va rămâne din construcție cu o înălțime mică. Acest tip de arbore se pretează la implementarea dicționarelor.

Un arbore B de ordin m este un arbore poziţional care are următoarele proprietăţi:

- Toate nodurile au cel mult m descendenţi;
- Toate nodurile cu excepţia rădăcinii au cel puţin m/2 descendenţi;
- Un nod care are *m* descendenţi notaţi cu D_0, D_1, ... , D_{m-1} are şi *m-1* chei notate cu K_1, K_2, ... , K_{m-1} cu proprietatea că $K_1 < K_2 < ... < K_{m-1}$.
- Pentru oricare nod cheia K_i este mai mare decât toate cheile descendentului notat D_{i-1} dar este mai mică decât toate cheile descendentului D_i. (cheile sunt situate între descendenţi)
- Toţi descendenţii NIL sunt situaţi la acelaşi nivel al arborelui.

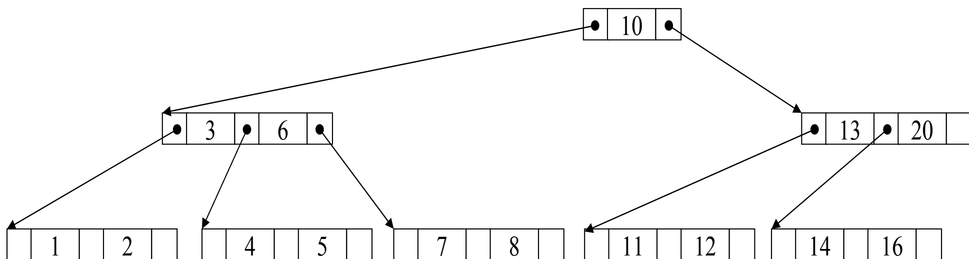


Fig. 24 Reprezentarea unui arbore B

2.11. BIBLIOGRAFIE

[35] Aho Alfred V., Jeffrey D. Ullman: *Data Structures and Algorithms*; Addison-Wesley, Reading, Massachusetts 1983.
[36] Brown Lawrence M.: *Introduction to Data Structures*; Nov. 2001.
[37] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L.: *Introduction to Algorithms*, McGraw-Hill, New York 1990.
[38] Eddon, Guy, and Henry Eddon, Inside Distributed COM, Microsoft Press 1998.
[39] Fox BL: *Data structures and computer science techniques in operations research*; Operations Research, Vol.26, Nr.5, Operations Research Society of America, September-October 1978.
[40] Goodrich Michael T., Roberto Tamassia: Data Structures and Algorithms in Java, John Wiley & Sons (1998).
[41] Ionescu Texe Clara, Zsako Ioan: *Structuri arborescente cu aplicaţiile lor*; Editura Tehnică, Bucureşti 1990.
[42] Joldos Marius: *Data Structures and Algorithms – Lecture Notes*; Technical University of Cluj-Napoca, Faculty of Automation and Computers, Computer Science Department, Cluj-Napoca 2002.
[43] Knuth, Donald. E.: *The Art of Computer Programming, Volume 3, Sorting and Searching*; Addison-Wesley, Reading, Massachusetts, 1998.

[44] Knuth, Donald. E.: *Tratat de programarea calculatoarelor. Algoritmi fundamentali*; Editura Tehnică, Bucureşti, 1974.
[45] Knuth, Donald. E.: *Tratat de programarea calculatoarelor. Sortare şi căutare*; Editura Tehnică, Bucureşti, 1976.
[46] Microsoft: *Microsoft Windows Architecture for Developers*; Course Number 794A, Microsoft Press 1997.
[47] Shaffer Clifford A.: *A Practical Introduction to Data Structures and Algorithm Analysis*, Java Edition, Prentice Hall 1998.
[48] Strohmeier Alfred: *Algorithms and Data Structures*; http://lglwww.epfl.ch, Swiss Federal Institute of Technology in Lausanne, Department of Computer Science, Software Engineering Laboratory, March 2000.
[49] Vogel Andreas, Keith Duddy: *Java Programming with CORBA*; Second Edition, John Wiley & Sons, New York 1998.
[50] Williams Peter M.: *Data Structures*; University of Sussex, UK, 2002.

Capitolul 3

BAZE DE DATE, PREZENTARE GENERALĂ

3.1. ISTORICUL BAZELOR DE DATE

Stocarea şi procesarea datelor nu este o invenţie şi o caracteristică a erei moderne. Din totdeauna omenirea a căutat să păstreze informaţiile pentru aducere aminte sau pentru posteritate. Iniţial pe scoarţa copacilor şi pe pereţii peşterilor apoi pe tăbliţe de lut, suluri de papirus, pergament, hârtie şi acum pe computere. Stocarea şi păstrarea sistematică a datelor, catalogarea, ordonarea şi regăsirea lor cu scopul prelucrării îşi are originea în vechile biblioteci şi arhive. Multe dintre principiile fundamentale ale bazelor de date au fost descoperite şi învăţate la acea vreme, ignorarea lor ducând la repetarea inutilă a unor greşeli de proiectare şi implementare.

Când calculatoarele electronice au devenit accesibile publicului s-a pus problema utilizării lor pentru stocarea de date. În anii 1960 s-au depus eforturi pentru a standardiza acest proces şi au rezultat două modele: modelul reţea (CODASYL) şi modelul ierarhic (IMS). Accesul la date se făcea încă într-un mod greoi prin operaţii de nivel scăzut. Adăugarea unui nou câmp impunea modificarea schemei de acces deoarece predominantă era preocuparea pentru definirea înregistrărilor şi nu structura sistemului. Erau necesare cunoştinţe de structură internă pentru scrierea programelor.

Cu toate aceste minusuri începutul era făcut şi procesul nu mai putea fi oprit. Astfel propunerea lui E.F. Codd din 1970 privind modelul relaţional al bazelor de date a venit firesc. El despărţea în două probleme distincte descrierea bazelor de date: organizarea logică şi metodele de stocare fizică.

Imediat (anii 1974-1977) propunerea teoretică a lui Codd a fost implementată practic în principal în două sisteme: Ingres dezvoltat de UCB cu limbajul de interogare QUEL şi System R dezvoltat de IBM la San Jose cu limbajul SEQUEL.

Următorul pas logic a fost dezvoltarea modelului Entitate-Relaţie pentru proiectarea bazelor de date. Această nouă abordare permitea concentrarea asupra utilizării datelor. Această propunere a fost făcută de P.Chen în 1976.

Combinând realizările anterioare Sequel 2 devine SQL şi în 1982 se începe standardizarea sa, proces încheiat în 1986. SQL devine un limbaj universal utilizat de foarte multe sisteme practice IBM DB2, RIM, RBASE 5000, PARADOX, OS/2 Database Manager, Dbase III şi IV, Watcom SQL şi numărul este mult mai larg.

După 1990 se merge pe dezvoltarea standardelor existente şi se pune un deosebit accent pe dezvoltarea de instrumente de acces la bazele de date. Direcţiile principale sunt: (1) posibilitatea de a crea programe utilizând limbajul de interogare şi (2) de a permite în limbaje de uz general includerea de comenzi de interogare. Totodată se începe definirea conceptului de sistemel de baze de date orientate pe obiect.

Tot în aceeaşi perioadă se distinge preocuparea creării de baze de date distribuite şi de utilizare a nou apărutei (la acea vreme) reţele globale Internet.

Implementările viitoare tind către utilizarea obiectelor ca element de bază şi de includerea conceptului de baze de date mobile în practica normală.

La mijlocul anilor 1990 are loc dezvoltarea explozivă Internet-ului. Se trece la introducerea aplicaţiilor client-server chiar pentru calculatoarele personale. Oferta de instrumente de gestionare a bazelor de date prin conectare la Internet creşte exponenţial. Active Server Pages, Front Page, Java Servlets, JDBC, Enterprise Java Beans, ColdFusion, Dream Weaver, Oracle Developer 2000, Apache, MySQL sunt doar câte exemple.

Un SGBD trebuie:

1. să permită utilizatorilor să creeze noi baze de date şi să le specifice schema (structura logică a datelor utilizând un limbaj specializat denumit *limbaj de definire a datelor*).
2. să ofere utilizatorilor capacitatea de a interoga datele (o „interogare" este argoticul pentru acţiunea de chestionare a datelor) şi de a le modifica, utilizând un limbaj adecvat, adesea denumit *limbaj de interogare* sau *limbaj de manipulare a datelor*.
3. să suporte stocarea unor cantităţi foarte mari de date pe perioade lungi de timp, ferindu-le de accidente sau de utilizare neautorizată şi permiţând acces eficient la date pentru interogare sau modificare.

4. să permită accesul simultan a mai multor utilizatori fără să permită ca acţiunile unuia să afecteze pe ceilalţi şi fără ca accesul simultan să ducă la coruperea accidentală a datelor.

Primele sisteme comerciale de baze de date au părut la sfârşitul anilor 1960. Aceste sisteme au evoluat din sisteme de fişiere şi furnizau parţial primele trei cerinţe. Ele erau mai degrabă grupuri de înregistrări. Primele modalităţi practice de a stoca şi regăsi date pe calculatoare erau aplicaţii pe fişiere care utilizau un limbaj de programare de nivel înalt (ex. COBOL). Principala deficienţă a acestei abordări era aceea că structura fişierului era descrisă în aplicaţie. Dacă cineva ar fi dorit să utilizeze datele într-o altă aplicaţie trebuia să scrie din nou descrierea fişierului în propriu său program. Mai mult, dacă din greşeală se pierdea sursa aplicaţiei înţelesul datelor s-ar fi pierdut. De aceea este necesar să se realizeze fişiere de date cu descrierea structurii ataşată sau inclusă. Pornind de aici s-au dezvoltat bazele de date.

Sistemele propriuzise de gestiune a bazelor de date au folosit la început diferite modele de descriere a structurilor cele mai populare fiind cea ierarhică sau arborescentă şi cea de tip graf sau reţea. ultima fiind standardizată la sfârşitul anilor 1960 prin raportul CODASYL (Committee on Data Systems and Languages). Problema esenţială a acestor modele şi sisteme era că nu incorporau şi nici nu se bazau pe un limbaj de nivel înalt pentru interogare.

Următorul pas la constituit a fost constituit de publicarea de către E.F. Codd a celebrelor sale lucrări din 1970 şi 1972 ([57], [58]). Aceste lucrări puneau bazele unei noi abordări a bazelor de date care erau organizate în tabele denumite relaţii. Pentru realizarea lor se puteau utiliza diferite implementări, structuri de date complexe şi mecanisme de stocare sofisticate, dar utilizatorul nu era preocupat de acest lucru. El gândea totul în termeni de relaţii. Interogarea se baza pe un aparat de lucru matematic asemănător celui din teoria mulţimilor ceea ce permitea exprimarea printr-un limbaj de nivel înalt. Lucrul acesta creştea eficienţa faţă de celelalte tipuri de date.

3.1. BAZE DE DATE COMPUTERIZATE

Pe computere iniţial omul a creat fişiere pe care le-a structurat pentru a uşura manevrarea. Prelucrarea datelor din fişiere, simplă şi eficientă, are însă numeroase limitări de aceea a apărut necesitatea realizării bazelor de date. Acestea nu sunt numai nişte structuri de fişiere mai exotice ci aduc cu ele o nouă modalitate de gândire despre date şi nevoile de exploatare.

Apar noi activităţi precum analiza datelor şi modelarea datelor care în timp au devenit dominante în proiectarea aplicaţiilor. Felurite facilităţi de verificare a integrităţii şi securităţii datelor au fost adăugate şi incluse într-un sistem de gestiune a bazelor de date care face posibil accesul utilizatorilor la resurse şi informaţii.

SISTEME DE PRELUCRARE A FIŞIERELOR

Aplicațiile bazate pe prelucrarea fişierelor, păstrează tot setul de date într-un fişier unic sau într-un set de fişiere. Datele care descriu sistemul sau componentele sale pot fi păstrate în fişiere separate şi este datoria softului pe care utilizatorul sau programatorul îl foloseşte să stabilească relațiile şi corespondența dintre fişiere.

Pentru fiecare fişier folosit programul trebuie să aibă inclusă permanent o interfață care să permită accesul la date. Mai multe fişiere înseamnă mai multe interfețe care să realizeze transferul. În cazul în care există mai multe aplicații care accesează aceleaşi fişiere interfețele trebuie incluse în fiecare aplicație în parte. Toate acestea duc la o "*explozie a interfețelor*" [54], programatorii trebuind să se ocupe mai mult de realizarea preluării datelor decât de problema de rezolvat.

Este de asemenea la latitudinea utilizatorului să selecteze numai date semnificative din fişier şi să „ascundă" înregistrările ce nu sunt necesare. Este, de aceea, foarte laborios să proiectezi şi să creezi un program (aplicație utilizator) şi este şi mai greu să o modifici în caz de necesitate. În multe cazuri, dacă din anumite motive, o anumită aplicație cerea modificarea formatului unui fişier toate celelalte aplicații trebuie să adopte un nou format de preluare a datelor deoarece interfața lor se realiza cu structura fizică a fişierului şi aceasta se modifica.

Din punct de vedere istoric, este primul sistem identificabil de baze de date şi încă se mai utilizează pe scară largă din cauza simplității, nevoii de compatibilitate cu versiunile anterioare de program şi din cauză că, în timp, sistemele de management al fişierelor a evoluat, putând răspunde la cerințe complexe.

Un avantaj este dat de faptul că diferite fişiere pot fi păstrate pe diferite calculatoare şi pot fi actualizate de persoane diferite. Dezavantajul major este dat de imposibilitatea de a păstra confidențial conținutul unui fişier. Securitatea informațiilor este serios afectată de posibilitatea de a citi direct fişierul şi a-l transporta pe un alt sistem.

Toate acestea au condus la necesitatea apariției unui nou mod de abordare a prelucrării datelor prin utilizarea bazelor de date.

SISTEME INTEGRATE DE PROCESAREA DATELOR

În acest tip de sisteme datele se stocau într-un singur loc şi structura internă a fişierului / fişierelor nu se mai documentează. Principalul pas înainte a fost introducerea unui Sistem de Gestiune a Datelor. Acest sistem permite doar comunicarea cu nucleul căruia i se transmite interogarea privind datele. Pentru aceasta au fost dezvoltate în timp limbaje şi proceduri complete pentru accesare.

Inițial sistemele de baze de date erau livrate ca atare numai cu mecanismele de acces şi de interogare. Era datoria utilizatorilor să-şi dezvolte aplicațiile. Ulterior, mai ales după apariția sistemelor de operare grafice de tip Windows, au fost furnizate

şi instrumente de construcţie a aplicaţiilor. Este acum extrem de uşor de dezvoltat, în manieră interactivă, forme de introducere sau de vizualizare a datelor.

Baza de date este o colecţie depozitată de date operaţionale utilizată de sistemul de aplicaţii al unei organizaţii.

Trebuie să facem unele precizări:

Organizaţie este un termen convenţional pentru a descrie la modul generic orice entitate comercială, ştiinţifică, tehnică sau operaţională.

Câteva exemple:

- Administraţia Naţională a Drumurilor,
- Compania Naţională a Căilor Ferate,
- Direcţie regională,
- Secţie,
- District,
- Universitate,
- Facultate,
- Institut de cercetări, etc.

Pentru buna funcţionare orice organizaţie trebuie să întreţină un set mare de baze de date. Acestea sunt baze date operaţionale. Datele operaţionale ale unei organizaţii de administrare a infrastructurii transporturilor pot fi: date de management; date de planificare; date de proiectare; date de climat; etc.

Se poate reduce redundanţa datelor stocate: În cele mai multe dintre sistemele actuale fiecare aplicaţie are fişiere proprii. Aceasta poate produce o utilizare ineficientă a spaţiului de stocare prin redundanţă. Utilizarea bazelor de date poate identifica şi elimina redundanţa.

Se pot evita problemele de inconsistenţă a datelor stocate: Din cauza existenţei surselor de eroare şi a transmiterii lor este posibil ca în fişiere diferite aceleaşi date să aibă valori diferite pentru a descrie acelaşi fapt. În bazele de date se va furniza aceeaşi valoare pentru orice interogare, indiferent de utilizator.

Datele pot fi partajate: Sistemele de baze de date oferă mecanisme prin care utilizatorii pot folosi în comun aceleaşi date simultan şi de a utiliza în comun aplicaţiile definite. În această manieră datele pot fi actualizate de oricare utilizator şi valorile prezentate celorlalţi pot fi în permanenţă cele reale.

Se pot aplica restricţii de securitate:

Administratorul bazei de date poate:

- Asigura că singurele căi de acces sunt cele autorizate prin canale , şi

- Defini verificări de autorizare pentru fiecare încercare de acces la date importante. Se pot defini proceduri diferite pentru fiecare tip de acces (citire, actualizare, ştergere etc.) pentru fiecare tip de date.

Poate fi menţinută integritatea datelor: Bazele de date pot trata mai eficient problema integrităţii. Chiar dacă redundanţa este eliminată se pot întâlni cazuri de date incorecte din cauza multor factori (eroare umană, defecte hardware, erori de programare etc.).

Cererile conflictuale pot fi rezolvate: Administratorul bazei de date poate stabili o importanţă pentru fiecare utilizator şi prin aceasta o prioritate cererilor de obţinere a datelor pe care le face.

Independenţa datelor: Fiecare utilizator sau aplicaţie poate avea propria viziune (VIEW) asupra datelor fără ca prin asta să afecteze .

Pentru a evita obligaţia utilizatorului de a calcula permanent şi repetat unele valori au fost concepute funcţii de tip TRIGGER care se declanşează la apariţia anumitor evenimente.

3.2 UTILIZAREA BAZELOR DE DATE

O **bază de date** reprezintă un ansamblu de date integrat, anume structurat şi dotat cu o descriere a acestei structuri.

Descrierea poartă numele de ***dicţionar de date*** sau ***metadate*** (informaţii despre date) şi creează o independenţă între datele propriu zise şi programe. O bază de date este mai mult decât o colecţie de fişiere: ea include, pe lângă acestea, dicţionarul de date şi o descriere a relaţiilor între înregistrări, descriere chemată şi utilizată pe întreaga durată a prelucrării informaţiilor.

De asemenea, vom numi o bază de date un ansamblu unitar organizat şi structurat de date, a căror gestionare se face printr-un sistem specializat, denumit sistem pentru gestionarea datelor (SGBD).

Des uzitată este şi noţiunea de **bancă de date** care reprezintă ansamblul format din:

- *bază de date*;
- *sistemul/sistemele* care o gestionează (SGBD);
- *echipamentele de calcul* utilizate pentru înregistrarea şi memorarea datelor din baza de date şi pentru prelucrările efectuate asupra acestor date;
- *procedurile* (automate şi neautomate) suplimentare pentru gestionarea datelor, în afara celor din cadrul SGBD, în primul rând pentru a face legătura / interfaţa cu acestea.

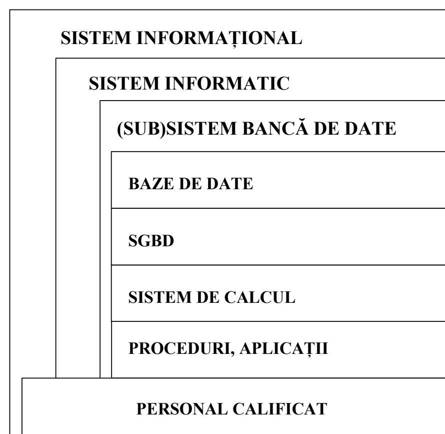


Fig. 25 Sistemul băncii de date

Spre deosebire de bazele de date care au mai mult un caracter personal, închis, banca de date este o entitate creată şi menţinută de o persoană (fizică sau juridică) ori de un compartiment specializat pentru a fi pusă la dispoziţia unui număr mare de alţi utilizatori, de exemplu publicul. Posedă în general o anvergură net superioară bazei de date, dar ceea ce o caracterizează cel mai bine este rigoarea necesară exploatării sale [53].

Se poate constata că banca de date are aceleaşi componente cu ale oricărui sistem informatic şi prin urmare ele vor trebui proiectate şi realizate după aceeaşi metodologie ca toate celelalte componente ale sistemelor informatice, folosind însă unele metode şi tehnici specifice.

În Fig. 25 se prezintă locul (sub)sistemului informatic *banca de date* în cadrul sistemului informaţional şi informatic al unei organizaţii.

Banca de date, în concepţia modernă de realizare a sistemelor informatice, devine subsistemul central al acestora, prin el realizându-se principalele legături dintre majoritatea celorlalte subsisteme şi aplicaţii, în primul rând a celor care lucrează cu datele din baza de date.

Dacă ne propunem să elaborăm un sistem informatic în care să utilizăm pentru gestionarea datelor conceptul de bază de date, atunci, este recomandabil să se facă acest lucru la toate sistemele unde se lucrează cu volume mari sau foarte mari de date.

Atunci când rezultă necesitatea utilizării unui SGBD, este recomandabil ca el să fie acelaşi pentru întregul sistem informatic al unităţii.

Facilitatea fundamentală oferită de abordarea prin prisma bazelor de date este că permite datelor să fie partajate între diferitele aplicaţii. Arhitectura generală a sistemului de baze de date are caracteristica de a aduna toate datele necesare într-un

fişier unic sau un set de fişiere. Seturile de date sunt separate numai *logic* în cadrul sistemului. Accesul la date se realizează prin **Sistemul de Gestiune a Bazei de Date** (SGBD) care este o însumare de proceduri software ce înţeleg şi manipulează structurile logice din fişier. SGBD pune la dispoziţia utilizatorilor facilităţi de stocare a datelor într-o structură adecvată, facilităţi de regăsire a lor, de protecţie, toate în condiţiile obţinerii unei bune performanţe. Ceea ce distinge cu adevărat un SGBD este posibilitatea de a descrie independent de reprezentarea lor fizică. În mod ideal, structura logică a unei baze de date este independentă de mediul de memorare.

Structura, pe straturi, a unui SGBD se compune din:

- nivel extern, văzut de utilizator, caracterizat de o anumită descriere a datelor, convenabilă acestuia;
- nivelul intern, transparent pentru utilizator, cu o altă descriere a datelor, conformă cu metodele de organizare şi acces puse la dispoziţie de ultimul nivel;
- sistemul de gestiune a fişierelor;

Deoarece aplicaţiile interacţionează numai cu sistemul de gestiune a bazei de date nu mai este necesară decât o singură interfaţă. Mai mult, această interfaţă se face la nivel logic şi nu fizic. De aceea aplicaţiile nu mai trebuie să cunoască structura intimă a fişierelor şi nici să-şi modifice modul de lucru de fiecare dată când se modifică structura datelor.

3.3 CARACTERISTICILE BAZELOR DE DATE

Un sistem de gestiune a bazelor de date îşi propune atingerea unor obiective care să definească astfel calitatea lui. Unele dintre aceste obiective sunt greu de atins, altele chiar contradictorii. Dar, se admite în general, că un sistem de gestiune a bazelor de date trebuie să asigure:

- independenţa fizică;
- independenţa logică;
- utilizarea datelor şi de către nespecialişti;
- eficacitatea accesului la date;
- administrarea centralizată a datelor;
- eliminarea redundanţei datelor;
- coerenţa datelor;
- partajarea datelor;
- securitatea datelor.

INDEPENDENŢA FIZICĂ

Unul dintre obiectivele importante ale unui SGBD îl constituie independenţa structurilor de memorare a datelor faţă de structura lor reală. Structurile de memorare trebuie să se supună regulilor de eficienţă a memorării şi regăsirii şi nu trebuie să fie

afectate de relaţiile existente între datele din lumea reală. Ele nu trebuie să fie imaginea organizării datelor din sistemul modelat. Altfel, o schimbare în structura externă a datelor ar necesita o reorganizare în structura internă, memorată a datelor.

INDEPENDENŢA LOGICĂ

Structura datelor din lumea reală, având o semantică proprie, reprezintă punctul de vedere al unui grup de utilizatori. Un alt grup poate avea o altă imagine asupra datelor, organizându-le altfel, sau considerând doar o parte a lor. Dacă un SGBD reuşeşte să ofere o asemenea structurare a datelor care să permită fiecărui grup de utilizatori să vadă datele aşa cum doreşte şi în acelaşi timp să permită grupurilor să-şi modifice punctul de vedere fără ca aceasta să afecteze alte viziuni particulare, atunci se poate spune că există o independenţă logică a datelor.

Împreună, independenţa fizică şi logică a datelor asigură stabilitatea aplicaţiilor faţă de modificările structurilor externe.

UTILIZAREA DATELOR DE CĂTRE NEINFORMATICIENI

Utilizarea de către nespecialişti a datelor stocate într-o bază de date înseamnă transformarea sistemului într-un instrument disponibil şi necesar cu o pătrundere într-o masă largă de utilizatori. Existenţa independenţei logice şi fizice creează posibilitatea ca un utilizator să vadă ceea ce doreşte din structurile bazei. Un limbaj neprocedural oferă utilizatorului neinformatician posibilitatea de a actualiza şi regăsi date fără a se preocupa de modul în care se realizează efectiv aceste operaţii.

EFICACITATEA ACCESULUI LA DATE

La baza de date au acces nu numai utilizatorii neinformaticieni ci şi specialiştii care asigură crearea şi întreţinerea acesteia. Primii folosesc un limbaj simplu, neprocedural, în spatele căruia se găsesc însă metode de acces evoluate şi o serie de primitive care să asigure construirea unor proceduri inteligibile. Toate acestea trebuie să satisfacă un standard de eficienţă pentru a face exploatarea posibilă, atât din punct de vedere al timpului de răspuns cât şi al calităţii şi varietăţii serviciilor oferite utilizatorului nespecialist. Astfel, dacă efortul de învăţare şi utilizare a limbajului de exploatare a bazei de date este excesiv, se pierde interesul pentru acesta.

Pe de altă parte utilizatorul specialist, administratorul bazei de date, trebuie să dispună de un limbaj de manipulare a datelor suficient de puternic şi flexibil pentru a avea acces uşor şi rapid la structurile cele mai intime, la structurile de memorare.

ADMINISTRAREA CENTRALIZATĂ A DATELOR

Administrarea datelor într-o bază de date constă în două operaţii principale: definirea structurii datelor şi modalităţilor de memorare şi urmărirea evoluţiei lor incluzând modificarea unor structuri. Având în vedere importanţa esenţială a acestei

funcţii (care în fond asigură integritatea şi acurateţea datelor) este preferabil ca ea să fie centralizată într-un singur punct, la dispoziţia unui grup de utilizatori privilegiaţi, şi să respecte un singur standard unitar.

ELIMINAREA REDUNDANŢEI DATELOR

În sistemele clasice aplicaţiile independente aveau nevoie de date aflate în fişiere proprii. Uneori aceleaşi date utile mai multor aplicaţii erau duplicate în tot atâtea fişiere rezultând de aici o utilizare ineficientă a spaţiului de memorare, dar şi dificultăţi în actualizarea corectă a tuturor datelor comune mai multor aplicaţii.

Bazele de date permit însă accesul partajat al mai multor aplicaţii la acelaşi fişier, eliminând astfel redundanţa. Gradul în care redundanţa este eliminată depinde de proiectarea corespunzătoare a structurilor de date. Totodată, administratorul bazei de date trebuie să vegheze ca actualizarea ulterioară să nu producă duplicare de date.

COERENŢA DATELOR

O condiţie esenţială a unui sistem de prelucrare utilizabil şi eficient este corectitudinea datelor manipulate. Proiectanţii, precum şi cei care exploatează un asemenea sistem trebuie să ia toate precauţiile pentru a sigura acurateţea datelor. Un sistem de gestiune a bazelor de date pune la dispoziţia utilizatorilor facilităţi de control a datelor introduse în bază.

Sunt verificate atât respectarea unor condiţii de validare la nivelul datelor elementare, cât şi al interdependenţelor între date în aşa fel încât coerenţa datelor să fie asigurată.

PARTAJAREA DATELOR

Aceasta presupune că ele sunt disponibile mai multor utilizatori. Probleme dificile apar în cazul simultaneităţii cererilor de acces la aceeaşi dată. Pentru rezolvarea unei asemenea concurenţe, există mecanisme la nivelul sistemelor de gestiune a fişierelor, care sunt preluate de sistemele de gestiune a bazelor de date.

SECURITATEA DATELOR

Datele trebuie protejate împotriva accesului neintenţionat sau răuvoitor. Mecanismele sistemului de gestiune a bazei de date verifică drepturile de acces ale utilizatorilor la toate datele, sau la o parte a acestora şi stabileşte modul de acces (scriere, citire, ştergere, actualizare).

3.4 MODELE DE BAZE DE DATE

În dezvoltarea sistemelor de gestiune a bazelor de date se disting mai multe generaţii.

Prima generaţie marchează epoca despărţirii SGBD de sistemele de gestiune a fişierelor. Este perioada primelor recomandări CODASYL când se realizează independenţa fizică şi logică. Eforturile principale se făceau în direcţia reducerii timpului de acces, arhitectura SGBD trecând pe planul secund.

A doua generaţie a fost fundamentată teoretic în anii '70 şi implementată în versiuni notabile în anii '80 şi este reprezentată de modelele relaţionale, care îşi propun uşurarea accesului la bazele de date prin crearea unei structuri externe cât mai simple şi mai naturale. Aceasta permite implementarea unor limbaje de manipulare foarte flexibile.

Modelele relaţionale cuprind un set de concepte şi reguli capabile să descrie datele şi acestea sunt văzute ca un set de relaţii asupra cărora se pot efectua anumite operaţii, definite de algebra relaţională, care au ca rezultat obţinerea unui nou set de relaţii.

Modelul relaţional manipulează structuri simple, liniare, neredundante. Complexitatea este dată de modul în care se combină flexibil aceste structuri. Conceptele de bază sunt: domeniu, tuplu, relaţie, normalizare.

Domeniul este o mulţime de valori legate semantic, apelabile sub acelaşi nume.

Considerând produsul a două sau mai multe domenii $D_1 \times D_2 \times \cdots \times D_n$ se obţin n–upluri sau *tupluri*. O seamă de tupluri astfel obţinute sunt contradictorii între ele. Eliminând convenabil aceste situaţii se obţine un număr de tupluri care reflectă o realitate. Submulţimea rezultată a produsului cartezian se numeşte relaţie.

Principiul de constituire a unei relaţii este cel al respectării unui fapt din realitate. Nu oricare două domenii pot constitui o relaţie. Asocierea lor trebui să constituie un tuplu semnificativ. O relaţie incorectă conţine redundanţe, contradicţii ori inconsistenţe. Într-o relaţie se poate defini o cheie primară care identifică în mod unic tuplul şi mai multe chei secundare. Cheia primară poate fi constituită dintr-un singur atribut fie prin concatenarea mai multor atribute.

Normalizarea se defineşte ca fiind operaţia de obţinere a unei relaţii corecte pe domeniul propus. Aceasta se face în mai multe etape (forme).

STRUCTURA BAZEI DE DATE

Pentru atingerea obiectivelor propuse anterior sunt propuse trei nivele de descriere a datelor:

- nivelul conceptual;
- nivelul intern;
- nivelul extern.

Nivelul conceptual corespunde structurii datelor din realitate, cu semantica lor, comună pentru toate grupurile de utilizatori. În lumea reală există date de tip elementar (atribute), care descriu un obiect şi care nu pot fi detaliate în continuare. La nivelul ierarhic imediat superior se află entitatea, obiectul real descris de atribute. Între aceste entităţi pot fi găsite relaţii, asocieri logice care reprezintă nivelul al treilea.

Proiectantul unui sistem trebuie să deceleze exact aceste elemente constituind modelul datelor şi interdependenţa lor. Acest model (schema conceptuală) cuprinde de asemenea şi informaţii referitoare la condiţiile pe care datele trebuie să le respecte.

Nivelul intern corespunde structurii de memorare a datelor (schema internă). La acest nivel se utilizează următoarele concepte:

- fişier/fişiere de date caracterizate prin nume, tip de organizare, localizare, dimensiuni;
- înregistrarea fişierului, definită de lungime, câmpuri, componente;
- căi de acces la înregistrare - prin indecşi, pointeri etc.

Nivelul extern este propriu fiecărui grup de utilizatori, care văd datele conform specificului aplicaţiei lor (schema externă).

O schemă externă vede doar o parte a bazei de date şi într-o structură proprie. Această particularitate are şi avantajul că oferă un mecanism în plus pentru asigurarea integrităţii şi securităţii datelor.

3.5. ARHITECTURA BAZELOR DE DATE

Grupul ANSI/X3/SPARC propune o arhitectură constând din două părţi principale:

- o parte care permite descrierea datelor;
- o parte care asigură manipularea datelor.

Structura

Procesorul schemei conceptuale, care permite descrierea schemei conceptuale a datelor, compilarea ei şi depunerea în metabază;

Procesorul schemei interne utilizat pentru descrierea schemei interne, a legăturilor ei cu nivelul conceptual, compilarea şi depunerea în metabază;

Transformatorul conceptual-intern utilizat pentru conversia între cele două nivele de descriere a datelor;

Transformatorul extern-conceptual care realizează manipularea datelor conform punctului de vedere al schemei externe;

Limbajul de manipulare a datelor externe care permite programatorului de aplicaţie să aibă acces la date conform cu punctul de vedere al schemei externe;

Interfaţa de acces la dicţionarul de date, prin intermediul căreia procesoarele şi celelalte elemente de arhitectură descrise deja, pot să aibă acces la scheme şi reguli de conversie compilate şi aflate în metabază.

Un exemplu de arhitectură este prezentat în figura următoare:

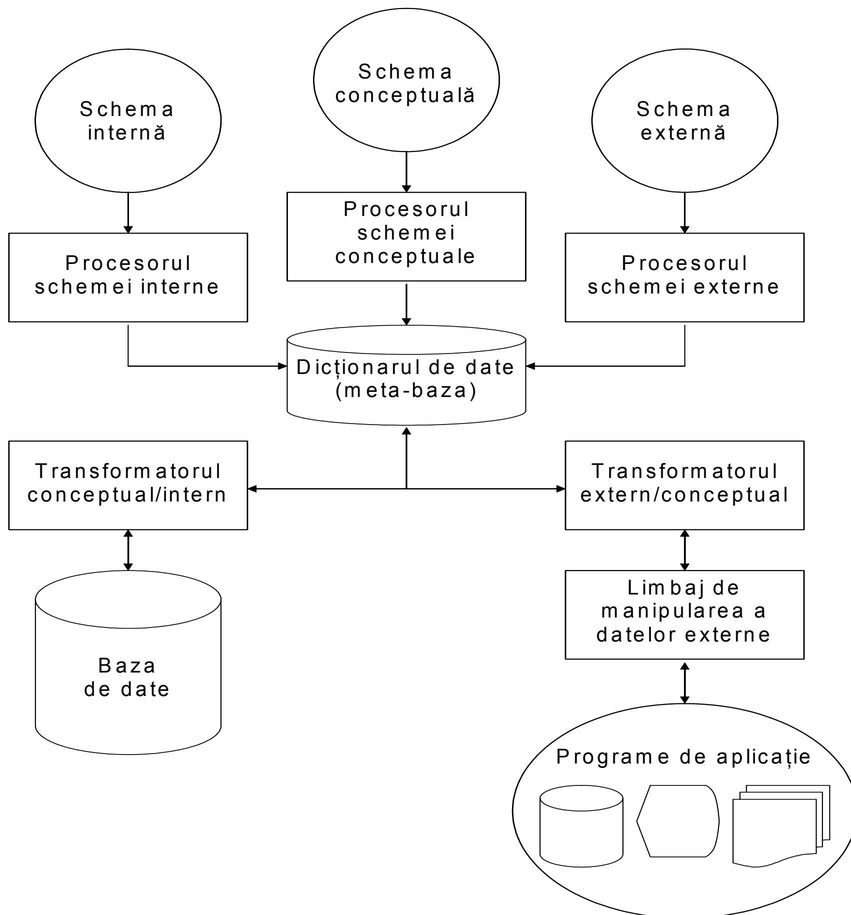


Fig. 26 Structura funcţională a unei baze de date

Tendinţele actuale de dezvoltare sunt de realizare a sistemelor de gestiune a bazelor de date distribuite şi cu procesare paralelă şi SGBD orientate pe obiecte. În general aceasta a apărut ca necesitate a decalajului semnificativ între evoluţia rapidă a tehnologiilor hardware şi de comunicaţie ce permit realizarea de medii eterogene complexe utilizând maşini tot mai performante şi realizările caselor de soft care susţin SGBD relaţionale.

3.6 CONCEPTE AVANSATE DE BAZE DE DATE

Denumirea de baze de date avansate este doar un apelativ generic. Ceea ce astăzi este avansat va fi perimat peste un deceniu sau două. Totuşi, sub acest nume se regăsesc ultimele tendințe din domeniu şi cea mai mare parte din munca universităților şi a diviziilor de cercetare. Deşi nu sunt clar definite şi separate tendințele interferează şi utilizează filosofii comune. Caracteristicile cele mai frecvent întâlnite sunt:

- Sisteme de gestiune cu procesare paralelă;
- Sisteme distribuite geografic;
- Baze de date orientate pe obiect;
- Capacități grafice.

Unele dintre aceste sisteme au influențe din domeniul inteligenței artificiale, la rândul ei aceasta fiind una dintre principalele aplicații.

SGBD CU PROCESARE PARALELĂ

Se bazează pe capacitatea procesoarelor de a oferi servere în multiprocesare simetrică(SMP), multiprocesare asimetrică sau procesare pe mai multe căi de control a execuției (*multithreaded*). Toate acestea implică rularea de task-uri multiple pe diferite procesoare. Diferența rezidă în faptul că dacă procesarea paralelă simetrică sau asimetrică implică mai multe procese ce se execută pe mai multe unități de prelucrare, multitheading are un singur proces ce se poate rula pe mai multe unități de prelucrare utilizând acelaşi context de mediu.

Tipuri de arhitecturi paralele

Cele mai utilizate arhitecturi hardware paralele ce exploatează multiple procesoare, memorii mari şi mai multe unități de disc sunt:

- arhitectura "*shared nothing*" în care fiecare procesor deține propriile sale unități de memorie şi de disc;
- arhitectura "*shared discs*" utilizează multiple procesoare, fiecare cu unitatea sa de memorie dar sus sistem de disc partajat.
- arhitectura "*symmetric multiprocessing*" utilizează procesoare multiple care dețin în comun unitățile de memorie şi de disc.

Sistemele dezvoltate pentru acest tip de calculatoare sunt în general extensii ale SGBD relaționale îmbunătățite pentru a beneficia de procesarea paralelă în sisteme eterogene şi de a permite rularea de aplicații complexe cu misiune critică. Ele utilizează o serie de noțiuni abstracte şi strategii asociate pentru a îndeplini cerințele actuale. De exemplu au fost create monitoare de procesare a tranzacțiilor ce reprezintă utilitare evoluate care gestionează tranzacțiile distribuite în rețele eterogene.

Din punctul de vedere al asigurării integrităţii datelor în sistemele client/server distribuite, producătorii de software de gestiune a bazelor de date au abordat următoarele strategii:

- tehnica "two-phase commit" prin care toate modificările impuse de o tranzacţie asupra unei baze de date sunt fie comise (execuţia tranzacţiei este finalizată) fie sunt anulate, cu revenirea bazei de date în stare anterioară efectuării tranzacţiei. Această strategie asigură faptul că toate serverele deţin copii identice în orice moment
- strategia de replicare a datelor este în prezent soluţia adoptată de marii producători. Replicarea constă în realizarea mai multor copii identice ale bazei de date. Garanţia identităţii este asigurată numai în anumite momente sau sub anumite condiţii.

BAZE DE DATE DISTRIBUITE

Un sistem distribuit implică existenţa unui număr de noduri de prelucrare în care există nişte colecţii locale de date ataşate fiecărui procesor.

Conceptual acestea pot fi privite, în context distribuit, ca părţi ale unei aceleiaşi colecţii de date, unică din punct de vedere logic, dar distribuită topologic.

Într-o bază de date distribuită spaţial fragmentele unei baze de date logice (baza de date aşa cum este văzută de utilizator) sunt în fapt câteva baze de date fizice fiecare fiind stocate pe diferite sisteme de calcul din diferite locaţii. În bazele de date distribuite de importanţă critică devine costul transmiterii datelor pe canalele de comunicaţie. Proiectanţii sistemului trebuie să minimizeze timpul de aşteptare a informaţiilor precum şi cantitatea de date care este transmisă efectiv de la o locaţie la alta în reţea.

Dispersia geografică şi autonomia colecţiilor de date sunt caracteristicile acestui tip de sisteme de gestiune a bazelor de date. Este necesar un software generalizat de gestiune care să permită facilităţi de organizare, acces şi control a datelor rezidente în diferite noduri de prelucrare interconectate fizic şi logic. Preluând funcţiile unui SGBD local, un sistem în context distribuit trebuie să asigure o bună funcţionalitate mai ales pentru următoarele activităţi:

- definirea unei arhitecturi în acord cu topologia sistemului;
- controlul prelucrării distribuite a informaţiilor din baza de date;
- controlul accesului concurent la datele distribuite;
- menţinerea integrităţii datelor în prezenţa defectelor hardware sau software;
- optimizarea timpului de răspuns al sistemului;
- menţinerea copiilor multiple ale bazei de date.

BAZE DE DATE ORIENTATE PE OBIECT

Modelele relaţionale sunt legate de abordarea bazată pe separarea datelor şi a prelucrărilor. Aceste modelele sunt în general inadecvate pentru construirea obiectelor din domenii cum ar fi CAD, administrarea documentelor, birotică, inginerie software, sisteme de informare, sisteme multimedia. În modelele relaţionale nu există noţiunile de clasă şi nici de obiect. Orice obiect ar trebui descompus în mai multe relaţii şi se stabilesc o sumă de legături pentru reconstituirea obiectului din părţi. O altă scădere este în diferenţa dintre limbajele de interogare şi manipulare şi limbajele de programare. Limbajele de programare şi limbajele de manipulare a bazelor de date sunt destul de dificil de integrat.

Pentru a rezolva aceste exigenţe au apărut două tendinţe.

Prima abordare constă în extinderea modelului relaţional. Extensiile integrează concepte precum agregare şi generalizare, noţiunea de clasă, seturi de atribute valorice, tipuri abstracte de date. De asemenea apare integrarea noţiunii de operatori ataşaţi unei proceduri. Operatorii sunt consideraţi ca atribute ale unei tabele şi conduc la definirea de noi tipuri de atribute.

În a doua abordare, ce se sprijină pe modele obiectuale, un obiect este considerat ca un tot unitar şi nu ca o mulţime de *n*-uple. Atributele unui obiect pot fi la rândul lor considerate obiecte. În linie generală, SGBDO integrează majoritatea funcţiunilor unui SGBD tradiţional şi în plus oferă un limbaj de programare orientat pe obiecte pentru a defini schema bazei, pentru manipularea obiectelor şi codificarea aplicaţiilor. Aceasta permite combinarea avantajelor programării orientate pe obiect cu conceptele de coerenţă, partajare, integritate proprii bazelor de date. Se reduce astfel la minim diferenţa dintre reprezentarea obiectelor la nivel conceptual şi nivel intern. Un obiect oricât de complex ar fi păstrează aceeaşi reprezentare în memorie ca şi pe disc ceea ce face ca să se elimine problemele de conversie de tip şi de structură. Legăturile dintre obiecte se păstrează la nivelul bazei de date.

Limbajele de programare orientate pe obiect s-au răspândit în mediile inteligenţei artificiale. Ele propun o nouă abordare a programării care îşi datorează originalitatea fuziunii naturale dintre prelucrare şi date.

Conceptul de bază este **obiectul**, pe care îl putem defini ca fiind reprezentarea unei entităţi a lumii reale. În baza de date obiectul este identificat într-o manieră unică printr-un identificator, numit OID (*Object IDentifier*), atribuit de sistem. Două obiecte având OID diferite sunt considerate ca diferite, spre deosebire de SGBD-urile relaţionale unde obiectele (tuplurile de valori corespunzând atributelor tabelei) nu sunt identificate decât prin valorile lor.

Principii de bază în orientarea pe obiect: Abstracţia, încapsularea, modularizarea, ierarhizarea.

Abstracţia

Orice model care include aspectele cele mai importante, esenţiale şi definitorii ale unor entităţi, suprimând sau ignorând detaliile mai puţin importante, imateriale, diversificatorii. Rezultatul eliminării distincţiilor este evidenţierea caracteristicilor comune.

În final caracteristicile abstracţiilor sunt „capturate” drept clase şi obiecte ale modelului.

Încapsularea

Localizarea fizică a facilităţilor într-o abstracţie de tip cutie neagră care le ascunde implementarea în spatele interfeţei publice.

Elimină dependenţa directă de implementare. Clienţii nu sunt afectaţi de modificarea implementării. Clienţii pot interacţiona numai prin interfaţă.

Modularizarea

Descompunerea logică şi fizică a lucrurilor în grupări simple care ajută la atingerea scopurilor de analiză.

Sparge sistemele în piese mai mici cu comportament mai simplu. Interfeţele dintre piese trebuie bine definite.

Ierarhizare

Orice ordonare sau aranjare într-o structură de tip arborescent. Aceasta corespunde organizării modelului dintr-un anumit punct de vedere.

Un obiect este descris prin ansamblul de proprietăţi constituit din **atribute** şi **metode**. Posibilitatea de a face private aceste proprietăţi vis-a-vis de obiect se numeşte **încapsulare**. Altfel spus, nici un alt obiect nu cunoaşte metodele unui obiect dat decât prin semnăturile lor (numele metodei, lista parametrilor şi tipul rezultatului returnat) şi nu are, în principiu, nici un mijloc de a accede la atributele acestuia, care sunt manipulate prin metode proprii.

Obiectele de aceeaşi natură sau prezentând similitudini sunt grupate în cadrul unei clase în care sunt declarate toate atributele şi metodele care le manipulează. O clasă este astfel un tipar pentru o mulţime de obiecte de acelaşi fel. Crearea unui nou obiect implică utilizarea instanţelor şi metodelor clasei din care aparţine.

Pentru a nu redefini aceleaşi atribute şi metode au fost create mecanisme de moştenire prin factorizarea proprietăţilor comune. Clasa nouă fiind nevoită să definească numai atributele şi metodele suplimentare.

Implementarea bazelor de date orientate pe obiect (BDOO)

BDOO combină conceptele orientării pe obiect, construcțiile programării şi posibilităţile gestionării bazelor de date. BDOO suportă partajarea concurentă şi referenţială şi netezeşte diferenţele dintre limbajele de programare şi bazele de date.

Orientare-pe-obiect = tipuri abstracte de date + moştenire + identitatea obiectelor

Proprietăţile bazelor de date preluate şi dezvoltate de BDOO sunt: persistenţă, tranzacţii, controlul concurenţei, recuperarea, tratarea cozilor de aşteptare, versiune, integritate, securitate şi performanţă.

1. *Persistența*: Capacitatea de a furniza aceleaşi date şi datele corecte în două tranzacţii diferite. Aceste date trebuie stocate pe un suport permanent.
2. *Tranzacția*: este o secvență de program care fie este executată în întregime fie nu este executată de loc. (aceasta se numeşte atomicitate);
3. *Controlul concurenței*: Sistemul furnizează o ordine în timp a tranzacţiilor utile (o ordine serială de execuţie);
4. *Recuperarea*: Trebuie să garanteze că rezultatele parţiale obţinute în urma unor actualizări incomplete desfăşurate în tranzacţii eronate nu se propagă în bazele de date persistente. Exemple de tipuri de eroare pot fi:

   Eşecul tranzacţiei (cauzat de conflicte între tranzacţii concurente);

   Defecte de sistem (eroare software în sistemul de operare, în SGBD sau cădere de tensiune);

   Defectarea dispozitivelor de stocare (uzual produse de căderea discurilor sau alte cauze).

5. *Interogare*: Se presupune existenţa unui limbaj de interogare şi a mecanismelor de acces. Utilizatorii pot specifica ***ce*** doresc fără să fie preocupaţi de ***cum*** sunt stocate fizic datele.
6. *Versiune*: Capacitatea de a oferi acces la stări anterioare ale obiectelor după ce au fost executate unele tranzacţii. Aceasta poate arăta evoluţia obiectelor.
7. *Restricții de integritate*: Bazele de date trebuie să permită accesul numai în stări stabile. Toate accesele către o zonă afectată de procese de tranzacţie trebuie blocate până când tranzacţia a fost corespunzător încheiată.
8. *Securitate*: trebuie definite o parolă şi drepturi de acces pentru fiecare utilizator.
9. *Probleme de performanță*: La momentul actual BDOO au reputaţia de a fi bogate în funcţionalitate dar slabe în performanţe ceea ce înseamnă că mai trebuie lucrat pentru a obţine o viteză ridicată.

TENDINŢE VIITOARE

Baze de date uriaşe vor necesita noi mijloace de manipulare şi analiză. Baze de date ştiinţifice foarte largi (de ordinul teraocteţilor) pentru explorarea spaţiului, securitate naţională, proiectul genomului uman, date de analiză geologică depăşesc bariere şi ridică noi probleme. Depozitele de date (data warehousing), cariere de date (data mining) şi târguri de date (data marts) sunt tehnici comune astăzi. Succesoare ale SQL, şi probabil ale SGBDR, vor fi dezvoltate pentru astfel de realizări. Cele mai multe dintre succesoare actuale ale SQL nu sunt încă utilizate la capacitate potenţială.

Bazele de date mobile sunt un nou produs care apare pe piaţă.

Viitorul aparţine bazelor de date inteligente. Bazele de date tind să încapsuleze cele mai noi concepte din domeniul tehnologiei informaţiei. Accesul la distanţă, sisteme de inteligenţă artificială, posibilităţi grafice şi hipermedia pentru prezentarea atractivă a informaţiilor.

3.7. AVANTAJE ŞI DEZAVANTAJE

Avantajele utilizării bazelor de date

- Se asigură controlul şi chiar eliminarea redundanţei;
- Consistenţa datelor;
- Mai multe informaţii din acelaşi set de date;
- Partajarea datelor;
- Îmbunătăţeşte integritatea datelor;
- Îmbunătăţeşte securitatea;
- Impune respectarea standardelor;
- Economie de scală;
- Echilibrează conflictele dintre interogări;
- Îmbunătăţeşte accesibilitatea şi disponibilitatea datelor;
- Productivitate crescută;
- Îmbunătăţeşte posibilitatea de întreţinere prin asigurarea independenţei datelor
- Îmbunătăţeşte concurenţa aplicaţiilor;
- Îmbunătăţeşte posibilităţile de salvare şi recuperare a datelor.
- Datele pot fi uşor partajate între aplicaţii, eliminând prin aceasta duplicarea şi probleme de întreţinere ce derivă din duplicare;
- Apeluri noi sau cereri de un anumit tip pot fi mult mai uşor implementate, datorită faptului că interfaţa logică cu SGBD este mai simplă decât un set de interfeţe fizice;
- Programele de aplicaţie sunt independente de datele stocate. Dacă formatul de stocare se modifică, nu este necesar să se modifice programele de aplicaţie, de vreme ce ele comunică logic cu SGBD;

- Se poate considera că un SDBD unic pentru o bază de date integrată permite o mai bună gestionare a datelor, de vreme ce ea este într-un singur loc sub supravegherea unui colectiv omogen;
- Integrarea şi partajarea datelor între aplicaţii dau posibilitatea unei programări sofisticate cu transmiterea de informaţii între utilizatorii bazei de date.

Dezavantajele utilizării bazelor de date

Ar fi necinstit să pretindem că bazele de date nu au dezavantaje:

- Sistemul de gestiune a bazelor de date şi echipamentul necesar pot fi scumpe şi pot avea preţuri de operare ridicate. Totuşi ele pot fi, în general, contrabalansate de o productivitatea înaltă.
- Se impun costuri ridicate pentru echipamentul hardware;
- Un SGBD este mult mai complex decât o prelucrare de fişiere şi are dimensiuni mari.
- În particular, recuperarea după o defecţiune poate fi dificilă. Un sistem de baze de date poate fi mai vulnerabil la defecţiuni decât un sistem de procesare a fişierelor. Impactul unei defecţiuni este mult mai mare datorită interacţiunii mai multor aplicaţii.

Este evident că trebuie făcută o analiză amănunţită şi o balansare între avantaje şi dezavantaje în contextul particular al mediului şi aplicaţiei considerate.

3.8. BIBLIOGRAFIE

[51] Astrahan M. M. et al.: System R: a relational approach to database management; ACM Trans. on Database Systems 1:2 (1976), pp. 97-137.
[52] Bernstein P. A. et al., "The Asilomar report on database research," http://s2k-ftp.cs.berkeley.edu:8000/postgres/papers/Asilomar_Final.htm .
[53] Blanc Pierre: *Initiation a l'nformatique*; Publié en libre service, Grenoble, 1995.
[54] Bowers DS: *From Data to Database*; Chapman and Hall, London, 1990.
[55] Carey M.J., D.J.DeWitt, G. Graefe, D.M. Haight, J.E. Richardson, D.H. Schuh, E.J. Shekita, S.L. Vandenberg: *The EXODUS Extensible DBMS Project: An Overview*; In Stan Zdonik and David Maier, editors, Readings In Object-Oriented Database Systems. Morgan-Kaufmann Publishers, Inc., 1990.
[56] Carey Michael J., DeWitt David J., Franklin Michael J., Hall Nancy E., McAuliffe Mark L., Naughton Jeffrey F., Schuh Daniel T., Solomon Marvin H., Tan C.K., Tsatalos Odysseas G., White Seth J., Zwilling Michael J.: Shoring Up Persistent Applications. In Proc. ACMSIGMOD International Conference on Management of Data, Minneapolis, pp.383–394, May 1994.
[57] Codd E. F.: *A Relational Model of Data for Large Shared Data Banks*; Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.
[58] Codd, E. F.: *Relational Completeness of Data Base Sublanguages*; in Data Base Systems. Vol. 6 of Courant Computer Symposia Series. Englewood Cliffs, N.J.: Prentice Hall, 1972. pp. 65--98.

[59] Comer Douglas: The Ubiquitous B-Tree; Computing Surveys, 11(2):121–137, June 1979
[60] Date, C. J., H. Darwen: *A Guide to the SQL Standard*; Fourth Edition, Addison-Wesley, Reading, MA, 1997.
[61] Guttman Antonin: *R-Trees: A Dynamic Index Structure For Spatial Searching*; In Proceedings of ACMSIGMOD International Conference on Management of Data, pp. 47–57, Boston, June 1984.
[62] Hellerstein Joseph M., Naughton Jeffrey F., Pfeffer Avi: *Generalized Search Trees for Database Systems*; Proceedings of the 21st VLDB Conference, Zurich, Switzerland, 1995.
[63] Kim Won, Kim Kyung-Chang, Dale Alfred: *Indexing Techniques for Object-Oriented Databases*; In Won Kim and Fred Lochovsky, editors, Object-Oriented Concepts, Databases, and Applications, pages 371–394. ACM Press and Addison-Wesley Publishing Co., 1989.
[64] Lin King-Ip, Jagadish H. V., Faloutsos Christos: *The TV-Tree: An Index Structure for High-Dimensional Data*; VLDB Journal, 3:517–542, October 1994.
[65] Lomet David B., Salzberg Betty. *The hBTree: A Multiattribute Indexing Method*; ACM Transactions on Database Systems, 15(4), December 1990.
[66] Mediano Maurício R., Casanova Marco A., Dreux Marcelo: V-Trees — A Storage Method For Long Vector Data; In Proc. 20th International Conference on Very Large Data Bases, pages 321–330, Santiago, September 1994.
[67] Parsaye Komran e.a.: *Intelligent Databases*; Wiley Press & Sons; 1989.
[68] Robinson J. T.: *The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes*; In Proc. ACM-SIGMOD International Conference on Management of Data, pp.10–18, Ann Arbor, April/May 1981.
[69] Stonebraker M., E. Wong, P. Kreps, G. Held: *The design and implementation of INGRES*; ACM Trans. on Database Systems 1:3 (1976), pp. 189-222.
[70] Stonebraker Michael: *Inclusion of New Types in Relational Database Systems*; In Proceedings of the IEEE Fourth International Conference on Data Engineering, pp. 262–269, Washington, D.C., February 1986.
[71] The POSTGRES Group: *POSTGRES Reference Manual*, Version 4.2. Technical Report M92/85, Electronics Research Laboratory, University of California, Berkeley, April 1994.
[72] Ullman Jeff D., Widom Jennifer: *A First Course in Database Systems*: Prentice-Hall, Englewood Cliffs NJ, 1997, 2nd edt. 2001.
[73] Wong C. K. , Easton M. C.: An Efficient Method for Weighted Sampling Without Replacement; SIAM Journal on Computing, 9(1):111–113, February 1980.

Capitolul 4

MODELE DE DATE UTILIZATE

4.1. DEFINIŢII

Înainte de a trece efectiv în revistă conţinutul acestui capitol trebuie să prezentăm câteva definiţii.

(N)-tuple reprezintă un set de (n) valori-atribut reprezentând o singură realizare a combinării domeniilor.

Atribut extern: este un atribut al unei entităţi care este importat, prin relaţiile stabilite, de la o altă entitate.

Atribut local: un atribut al unei entităţi ce este definit din proprietăţile intrinseci ale acelei entităţi.

Atribut: o funcţie pe un domeniu pentru fiecare realizare a tuplelor. Atribuie o singură valoare (dintr-o mulţime bine definită) fiecărui individ din mulţimea entităţilor. Este o caracteristică a entităţii care se selectează pentru a fi inclusă în modelul E-R.

Atribute surogat: un atribut al unui tip de entitate care este conceput de proiectantul bazei de date fără legătură cu regulile semantice care definesc entitatea.

Bază de date: un set de relaţii.

Cardinalitate: numărul de tuple pe care le are o relaţie. Este o restricţie asupra relaţiei care descrie numărul maxim de indivizi dintr-o set activ ce poate fi corelat cu un singur individ dintr-un alt set activ. Cardinalitatea poate fi unul-la-mai-mulţi, mai-mulţi-la-unul, unul-la-unul.

Cheie (candidat): o (super)cheie care este *minimală* (care nu mai are un subset corespunzător care să identifice unic fiecare tuplu din relaţie). Pot exista mai multe pentru fiecare relaţie.

Cheie (supercheie): un set de atribute a cărui valori grupate identifică *unic* fiecare tuplu din relație.

Cheie externă: este o cheie candidat a unei relații situată într-o altă relație.

Cheie primară: o cheie candidat aleasă a fi cheia principală a unei relații. Pentru o relație există o singură cheie primară.

Colecție: este un set de înregistrări, fiecare cu acelaşi număr şi tip de elemente de date.

Domeniul: mulțimea tuturor valorilor posibile ce corespund unui atribut şi care pot fi stocate. Un domeniu are un format şi un tip.

Element al entității: un obiect specific (obiect fizic sau conceptual, eveniment, faptă) asociat cu activitatea.

Elemente de date: fiecare dintre valorile elementare care sunt stocate în bazele de date. Fiecare dintre elementele de date are tip de dată (pe scut tip) şi pot fi grupate într-o înregistrare.

Entitate: este oricare „ceva” care este parte a problemei şi care trebuie reprezentat în baza de date.

Gradul: este numărul de atribute pe care le are o relație;

Înregistrare: un set de unul sau mai multe elemente de date cu nume propriu.

Legătură: împerecherea unui individ dintr-un set activ cu un individ dintr-un alt set activ.

Mapare: setul tuturor legăturilor dintre membrii individuali ai seturilor active.

NULL: o valoare constantă specială, compatibilă cu domeniul oricărui atribut care denotă faptul că atributul este nedefinit pentru un element al entității.

Produs cartezian: Produsul cartezian (·) între mulțimi este mulțimea tuturor combinațiilor posibile între elementele seturilor.

Relație: un subset al produsului cartezian al domeniilor sale.

Rolul: mai multe atribute pot avea acelaşi domeniu, dar atributele indică rolul diferit în relație.

Schema relațională: o descriere a conexiunii dintre două tipuri de entități conformă cu maparea membrilor unui set activ la membrii altui set activ.

Setul activ (al unui tip de entitate): toate acele entități individuale de un anumit tip care participă la momentul respectiv la o activitate.

Tipul entității: o clasă de individualități asociată cu acțiunea, care poate fi descrisă de un set finit de reguli sintactice şi semantice determinate de acțiune – pe baza proprietăților lor reale, intrinseci şi a rolului lor în acțiune.

Valoare-atribut: rezultatul funcției atribut. Fiecare realizare este reprezentată de o valoare-atribut extrasă din fiecare domeniu şi o valoare specială NULL.

4.2. MODELE DE DATE

Un model de date este o colecție de concepte care descriu datele astfel încât ele să poată fi asociate şi incluse într-o bază de date.

Într-un mod mai formal, un model de date este o combinație a cel puțin trei componente:

- O colecție de tipuri de structuri de date;
- O colecție de operatori sau reguli de raționament care pot fi aplicate oricărei instanțe valide a tipurilor de date;
- O colecție de reguli generale de integritate care definesc implicit sau explicit mulțimea stărilor bazei de date, posibilitățile de modificare sau amândouă.

Este importat de observat că modelul de date este o construcție logică care se realizează ulterior pe diferite platforme hardware şi software. De aceea, acelaşi model de date poate avea diferite aparențe, performanțe, eficiență, limite, costuri etc. funcție de implementare.

MODELUL ENTITATE-RELAȚIE

Cel mai răspândit model de reprezentare abstractă a structurii bazelor de date este modelul entitate–relație (model E/R). În acest model structura datelor este reprezentată grafic printr-o „diagramă entitate-relație".[79]

Modelul E/R implică utilizarea a trei elemente: entitatea, atributul şi relația.

Entitatea

Entitatea este „ceva" care este parte a problemei şi care trebuie reprezentat în baza de date. Poate fi un obiect fizic (o persoană, o clădire, un tunel etc.), conceptual (o instituție, un proiect etc.) sau faptic (o revizie periodică a fost efectuată şi au fost efectuate măsurători care trebuie stocate). În sens larg, o entitate este un obiect abstract de orice natură. O colecție de entități similare formează un set de entități.

Există asemănări între entitate, aşa cum a fost prezentată mai sus, şi termenul de obiect din programarea orientată pe obiect. Totuşi, modelul E/R este un concept static implicând doar structuri de date nu şi operațiile asupra datelor (Ullman şi Widom 2001 [84]).

Pentru termenul ***entitate*** putem să găsim mai multe semnificații care rezultă din context. Dacă există posibilitatea de confuzie înțelesul trebuie specificat în clar.

O ***entitate individuală*** este un lucru specific (obiect fizic sau conceptual, acțiune sau eveniment) asociat cu un fapt. Referirea la o entitate individuală se face cu articol (hotărât sau nehotărât).

Un ***tip*** de entitate este o clasă de entități individuale care pot fi descrise colectiv printr-un set finit de reguli sintactice şi semantice determinate de necesitățile specifice ale administratorilor şi utilizatorilor. Acestea sunt prototipuri care definesc indivizii. În diagramele E-R sunt reprezentate tipurile de entitate.

Referirea se face nearticulat sau la plural.

Setul activ al tipului de entitate reprezintă secţiunea din mulţimea totală a indivizilor care participă la un realizare. În modelul E-R nu suntem interesaţi de setul activ decât în execuţie, acesta fiind rezultatul aplicării unei interogări.

Referirea se face utilizând pluralul.

De notat că entităţile individuale, tipurile de entitate şi seturile active există efectiv în realitate dar nu pot fi stocate direct în baza de date. Trebuie să identificăm caracteristicile care definesc fiecare entitate.

Atributele

În cadrul modelului, entităţile trebuie caracterizate pentru a se diferenţia din setul de entităţi. Aceasta se face prin proprietăţi care se ataşează entităţilor şi poartă denumirea de ***atribute***. Atributele au, în general, valori atomice: întregi, reali, şiruri de caractere. Există şi variante ale modelului E/R în care atributele au formă structurată.

În proiectarea unei baze de date trebuie identificate caracteristicile esenţiale ale unei entităţi şi mulţimea valorilor care pot fi acordate acestor caracteristici. Noţiunea de „esenţial" se referă la contextul specific în care bazele de date şi aplicaţiile aferente vor fi utilizate.

De exemplu, când se realizează o listă a personalului care lucrează în administraţia drumurilor este total irelevant să stocăm date privind culoarea părului, înălţime, culoarea ochilor etc. pe când un sistem de baze de date al poliţiei consideră tocmai aceste date de importanţă deosebită.

Spre deosebire de entităţi, atributele pot fi stocate în computer. Ele sunt definite ca valori stocabile, cuprinse într-un anumit domeniu.

Exemple

- Persoanele angajate au nume, prenume, funcţie, studii, adresă, telefon, competenţe etc.

- Sectoarele rutiere au poziţie, lăţime, material de construcţie, pantă longitudinală, pantă transversală, acostament etc.

- Podurile au poziţie, lungime, lăţime, număr de benzi, număr de deschideri etc.

Relaţiile

Relaţiile sunt conexiuni care se stabilesc între entităţi. Relaţia poate apare între două entităţi (***relaţie binară***) dar este posibilă relaţia între oricâte entităţi.

Relaţiile reprezintă o hartă a legăturilor dintre membrii fiecărui set de entităţi.

În fapt există mai multe niveluri de noţiuni legate de conceptul de relaţie.

Tabelul 1. Noţiunea de entitate

Conceptul de entitate	**Concepte relaţionale**
Tip de entitate	Relaţie
Set activ	Harta conexiunilor
Entităţi individuale	Legături

Restricţiile numerice ale relaţiilor

Relaţia dintre două tipuri de entităţi se caracterizează prin restricţii legate de dimensiunea maximă a seturilor active care pot participa (restricţii de cardinalitate). Pot exista relaţii:

- **Unul la unul** – când există o legătură strictă biunivocă între un sigur individ din fiecare tip de entitate;
- **Unul la mai mulţi** – unui individ din primul tip de entitate îi corespund mai mulţi membri din al doilea tip de entitate;
- **Mai mulţi la unul** – mai mulţi membri ai primului tip de entitate sunt legaţi de un membru al celui de al doilea tip de entitate.

Reprezentarea restricţiei de cardinalitate se face convenţional după cum se prezintă în figura următoare:

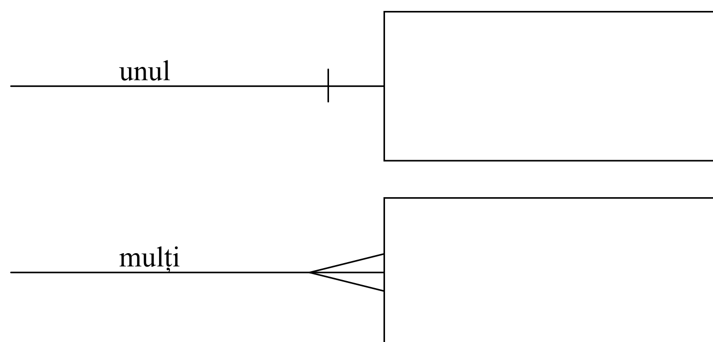


Fig. 27 Reprezentare restricţiilor de cardinalitate

Diagrame entitate relaţie

O diagramă entitate-relaţie (diagramă E/R) este un graf care reprezintă seturi de entităţi, atribute şi relaţii. Elementele din fiecare sunt reprezentate de nodurile

grafului. Pentru fiecare tip de element se utilizează reprezentarea prin forme speciale predefinite, stabilite prin convenţie.

Cele mai utilizate reprezentări convenţionale sunt:

- Setul de entităţi este reprezentat prin dreptunghiuri;
- Atributele sunt reprezentate prin elipse;
- Relaţiile se reprezintă prin romburi

Laturile grafului conectează o entitate de atributele sate şi seturile de entităţi de relaţiile desemnate.

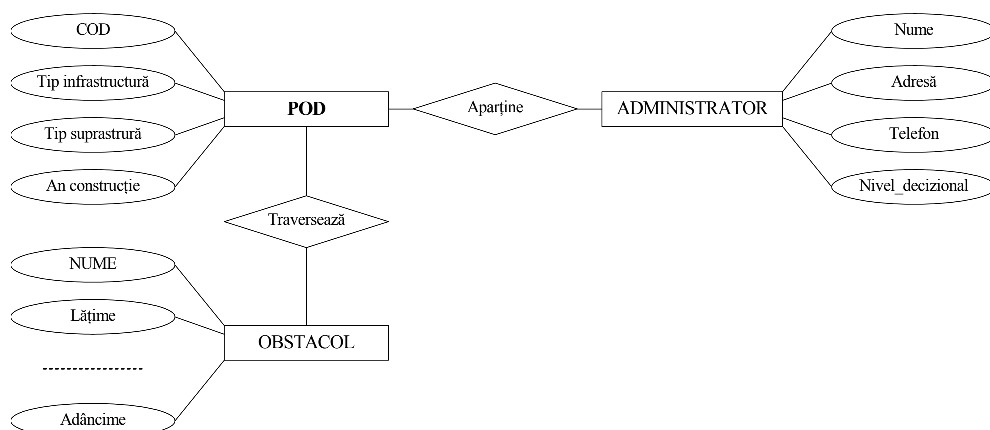


Fig. 28 Reprezentare Entitate – Relaţie

În figura de mai sus am prezentat un exemplu de diagramă entitate-relaţie simplă pentru trei seturi de entităţi, fiecare cu atributele sale. Seturile de entităţi sunt conectate de relaţiile specifice. Astfel podurile *traversează* obstacole şi tot podurile *aparţin* unor administratori.

Atribute ale relaţiei

Uneori este imposibil sau extrem de dificil a ataşa anumite atribute cu un set de entităţi. În astfel de situaţii este convenabil a ataşa atribute la relaţie.

Să consideră, de exemplu, că se stabileşte un contract pentru realizarea lucrărilor de reabilitare ale unei lucrări de artă. La contract participă executantul lucrărilor desemnat drept „constructor" şi gestionarul lucrării de artă – „administratorul".

Deoarece la realizarea reabilitării pot participa mai mulţi executanţi contractuali nu putem ataşa preţul contractului la entitatea numită „reabilitare".

Constructorul poate participa la executarea mai multor contracte simultan, iar administratorul, la rândul său, gestionează mai multe activități. Nu este convenabil să ataşăm atributul „valoare negociată” de nici-unul dintre seturile de entități. Locul său este lângă relația numită „contract”.

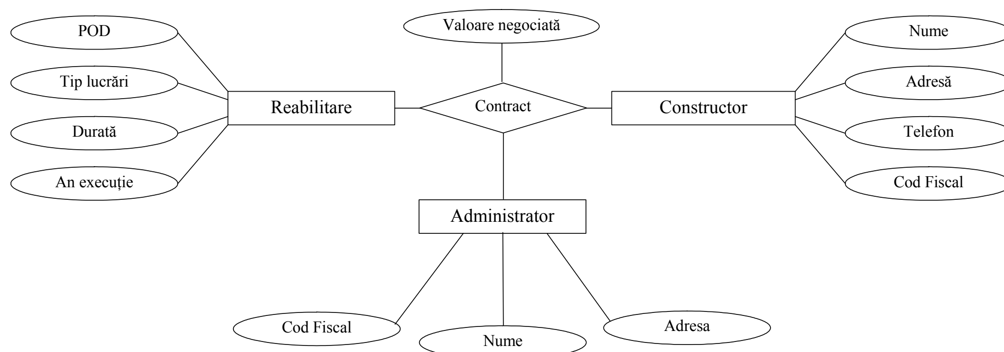


Fig. 29 Reprezentare Entitate – Relație pentru atribute ataşate relației

Chiar dacă uşor de manipulat, acest concept nu este absolut necesar. Se poate crea o nouă entitate căreia i se ataşează atributul necesar şi se adaugă entitatea în relație.

Reprezentarea atributelor

Înaintea definitivării modelului, trebuie să identificăm atributele care reprezintă relația. Totodată trebuie să stabilim motivația şi consecințele alegerii unui atribut.

Exemplu: Presupunem că un constructor încheie contracte pentru lucrări. Pe perioada construcției, atributele ataşate constructorului devin atribute ataşate lucrării.

Deosebim trei tipuri de atribute:

- locale,
- surogat şi
- străine.

Atributele locale: sunt atributele intrinsec definite tipului de entitate. Aceasta înseamnă că indiferent de relațiile în care este implicată entitatea atributul este prezent şi are o valoare bine definită. Numele este un atribut local al constructorului.

Atributul surogat: reprezintă un atribut al tipului de entitate care a fost inventat pentru o mai bună manipulare a datelor. Codul numeric personal pentru angajați, codul fiscal pentru societăți sau un număr unic de identificare acordat automat de sistem pot fi considerate atribute surogat. Ele reprezintă doar convenții fără corespondent în realitate dar asigură o mai bună distincție între indivizi.

Atribut străin: reprezintă un atribut al unei entităţi care este ataşat unei alte entităţi aflate în relaţie cu prima pe durata relaţiei. Dacă avem contracte de lucrări efectuate de un constructor, pe perioada relaţiei atributele care îl identifică pe constructor devin atribute ale contractului.

4.3. DEZVOLTAREA BAZELOR DE DATE

NIVELE ALE ARHITECTURII BAZELOR DE DATE

Există trei nivele ale arhitecturii bazelor de date:

- Nivelul conceptual;
- Nivelul logic;
- Nivelul fizic.

1. *Nivelul conceptual*: defineşte principiile structurii (fişiere, reţele, pointeri, tabele relaţionale etc.). Aceasta este doar o abstractizare a lumii reale care se concentrează asupra elementelor de informaţie care sunt relevante pentru utilizatorii bazei de date.
2. *Nivelul logic*: Structurile bazei de date sunt definite la nivelul schemelor. Schemele includ o descriere a tipurilor de înregistrări şi a modurilor în care sunt corelate.
   a) *Dezvoltarea de scheme şi dicţionare de date*: Trebuie identificate relaţiile şi atributele şi se cataloghează în dicţionare de date. Dicţionarul de date este, el însuşi, o bază de date constând din conceptele esenţiale pe care ar trebui să le cunoască utilizatorii care lucrează cu baza de date. Funcţiile furnizate de dicţionar sunt:
      (i) Definirea câmpurilor şi fişierelor;
      (ii) Definirea relaţiilor;
      (iii) Definirea schemelor şi sub-schemelor externe (cum pot fi accesate datele, lista câmpurilor, lista relaţiilor etc.)
      (iv) Definirea elementelor fizice importante: localizarea datelor, volum etc.
      (v) Definirea viziunilor.
   b) *Scheme de ierarhie şi reţea*: stabileşte legături între înregistrările din baza de date. Aceasta se ilustrează într-un mod specific. Sistemul trebuie să includă:
      (i) Declaraţia numelor de scheme;
      (ii) Una sau mai multe declaraţii ale tipurilor de înregistrare;
      (iii) Una sau mai multe declaraţii de seturi care definesc relaţiile dintre tipurile de înregistrări;
      (iv) Una sau mai multe declaraţii ce definesc zonele fizice în care se stochează înregistrările.

c) *Scheme relaționale*: furnizează un model de a privi datele. Utilizând algebra relațională datele pot fi reunite, proiectate, selectate etc.

3. *Nivel fizic*: abordează probleme specifice implementării. Include accesarea şi metodele de management a datelor (secvențial, indexat etc.).

MODELE DE BAZE DE DATE

În dezvoltarea bazelor de date s-au utilizat mai multe modele:

- Modelul ierarhic,
- Modelul rețea,
- Modelul relațional.

Modelul ierarhic

Modelul a fost dezvoltat în anii '60 la North American Rockwell.

Acest model se referă la aranjarea ordonată datelor într-o structură de forma unui arbore cu rădăcina în sus. Un exemplu este cel prezentat în figura următoare:

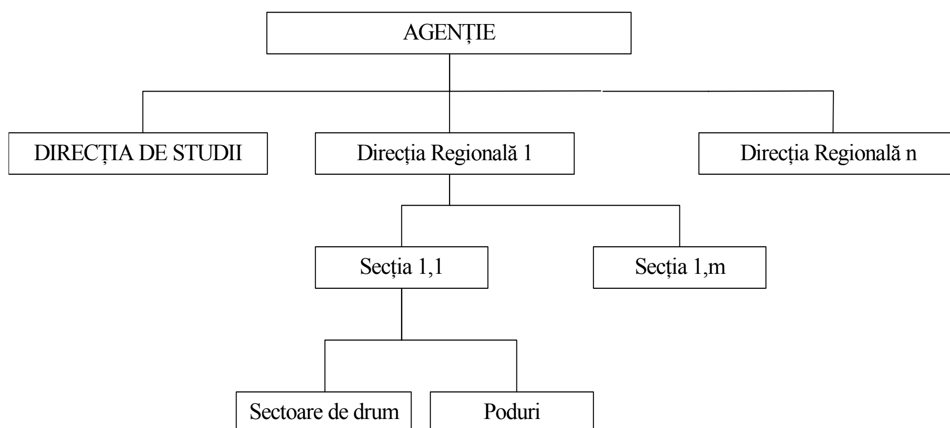


Fig. 30 Modelul ierarhic pentru bazele de date

Aşa cum se poate vedea mai sus, înregistrările din acest model respectă o ierarhie strictă. Legăturile dintre diferitele structuri se fac prin pointeri. Un pointer este o referință care indică localizarea unor date specifice în mediul de stocare. Intuitiv, pointerii sunt precum referințele de pagină în zona de index dintr-o carte.

Modelul ierarhic a fost prima implementare comercială majoră a conceptelor teoretice despre bazele de date. Caracteristica de bază a structurii datelor în acest model este prezența în percepția utilizatorilor a diferitelor niveluri. Cel mai de sus nivel este rădăcina sau părintele.

Parcurgerea bazei de date se face respectând calea ierarhică. Pentru a ajunge la un anumit segment de date trebuie parcursă întreaga cale de la rădăcină la segmentul specific.

Modelul rețea

Modelul rețea a fost creat în 1971 de Conferința asupra Limbajelor Sistemelor de Date (Conference on Data Systems Languages - CODASYL). Simplificat, modelul de baze de date rețea este prezentat într-o manieră intuitivă în figura următoare:

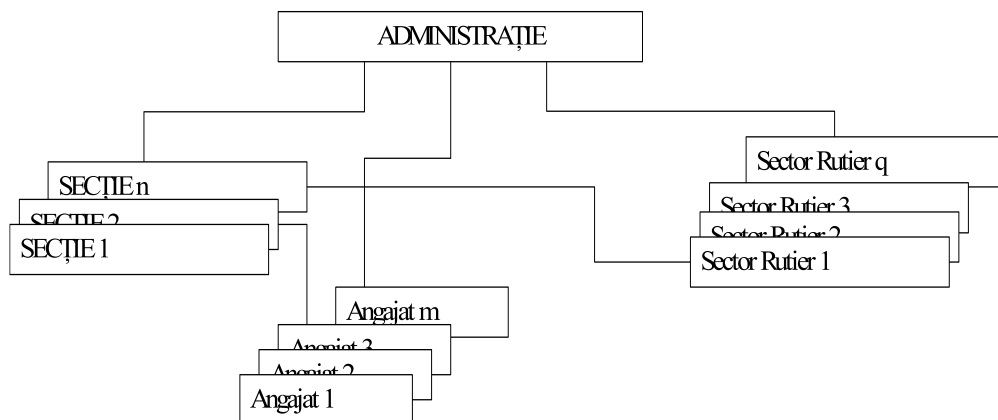


Fig. 31 Modelul de baze de date rețea

Structura de bază în acest model este setul şi se compune din cel puțin două tipuri de înregistrări: o înregistrare posesor echivalentă cu părintele din modelul ierarhic şi o înregistrare membru. Un membru poate avea mai mulți posesori.

Modelul de date relațional

Modelul relațional a fost dezvoltat în 1970 la IBM de E.F. Codd. La acea vreme modelul a fost considerat ingenios dar nepractic. Dezvoltarea ulterioară a calculatoarelor ca şi scăderea prețurilor a permis ca acest model să fie utilizat de sisteme de baze de date sofisticate: ORACLE, Informix, Ingress etc.

Structura de bază este tabela. Fiecare tabelă este o matrice formată din linii şi coloane. Tabelele sunt referite prin relații. Aceasta înseamnă că tabelele conțin în comun unele caracteristici pe baza cărora se pot stabili legături.

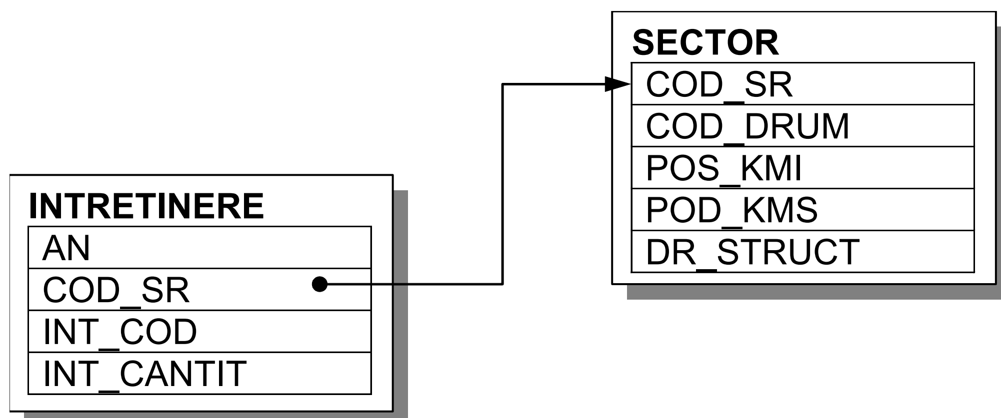


Fig. 32 Modelul relațional de baze de date

Probabil cea mai simplă formă de abordare a modelului relațional rămâne cel propus inițial de Codd, chiar după rafinări şi dezvoltări succesive. Modelul relațional specifică o structură de date numită relație şi câteva expresii de manipulare a relațiilor. Termenul de relație reprezintă o tabelă, structurată pe linii şi coloane, în care se aranjează datele. Nu este obligatoriu ca datele să fie omogene. Pentru relații a fost definit şi un set de operații denumit algebră relațională care va fi prezentat în capitolul următor.

Considerații practice

Modelele de baze de date au fost utilizate ca fundamente teoretice pentru dezvoltarea a bazelor de date şi a sistemelor de gestiune a bazelor de date. În ultima perioadă cea mai mare extindere au avut-o bazele de date relaționale care au fost susținute puternic de industria informaticii dar şi de utilizatori care au găsit în acest model un răspuns simplu la necesitățile lor.

Este motivul pentru care, în continuare vom dezbate doar problemele legate de modele de date relaționale, problemele teoretice şi dezvoltarea limbajelor aferente insistând pe dezvoltarea limbajului SQL ca standard în domeniul bazelor de date relaționale.

4.4. BIBLIOGRAFIE

[74] Chen, 1976: Chen Peter P.: *The Entity-Relationship Model - Toward a Unified View of Data*. ACM TODS 1(1): 9-36(1976)

[75] Codd, 1970: Codd E. F.: *A Relational Model of Data for Large Shared Data Banks*; Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[76] Codd, 1972: Codd, E. F.: *Relational Completeness of Data Base Sublanguages*; in Data Base Systems. Vol. 6 of Courant Computer Symposia Series. Englewood Cliffs, N.J.: Prentice Hall, 1972. pp. 65--98.

[77] Codd, 1979: Codd E. F. (1979): *Extending the Data Base Relational Model to Capture More Meaning*; ACM Trans. on Database Systems, 4, 4, 397-434. 48
[78] Codd, 1981: E. F.: *The 1981 ACM Turing Award Lecture: Relational Databases – A practical foundation for Productivity*; CACM, Vol. 25 No 2, February 1982
[79] Dick Wayne, Jewett Tom: Practical Relational Database Design, 1999 ????
[80] Garcia-Molina Hector, Ullman Jeffrey D., Widom Jennifer D.: *Database Systems: The Complete Book*; Prentice Hall, Copyright: 2002.
[81] Muller Robert J.: *Database Design for Smarties: Using UML for Data Modelling*; Morgan Kaufmann, 1999.
[82] Muller Robert J.: *Database Design for Smarties: Using UML for Data Modelling*; Morgan Kaufmann, 1999.
[83] Tsichritzis, 1982: Tsichritzis, F.H. Lochovsky: *Data Models*; Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
[84] Ullman Jeffrey D., Widom Jennifer D: *A First Course in Database systems*, (First edition 1997, second edition published in October, 2001.) Prentice Hall, 2002.

Capitolul 5

ALGEBRA RELAŢIONALĂ ŞI LIMBAJE DE BAZE DE DATE

5.1. ALGEBRA RELAŢIONALĂ

În cadrul SGBD procesarea datelor se poate face prin limbaje de manipulare a datelor (DML). DML pot fi diferite funcţie de sintaxa lor dar mai ale funcţie de operaţiile puse la dispoziţie. Aceste operaţii, materializate prin definirea unui operator, fundamentează complexitatea şi puterea de prelucrare a limbajului. Mulţimea operatorilor definiţi formează algebra.

Algebra relaţională este un set de operatori definit pe mulţimea relaţiilor şi care dau ca rezultat o relaţie. Prin utilizarea lor se pot defini interogări complicate.

Se definesc 6 operatori de bază:

- Proiecţia (π),
- Selecţia (σ),
- Produsul cartezian (×),
- Uniunea ($\bigcup$),
- Diferenţa (–) şi
- Redenumirea.

Proiecţia

Proiecţia se notează cu $\pi_{a_1,a_2,\cdots,a_m}(r)$ unde *r* este relaţia asupra căreia se aplică operatorul iar $a_1, a_2, \cdots, a_m$ reprezintă atribute. Relaţia rezultată prin aplicarea

operatorului de proiecție are *m* coloane extrase din relația *r*. Atributele (coloanele) din *r* care nu sunt enumerate nu se păstrează. Duplicatele nu se rețin.

Exemplu: Se consideră o relație *r* de forma:

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	12	6	200
DN	1	78	555
DJ	105	1	000
DJ	105	1	780
DN	1	13	762
DJ	1	45	300

Definim proiecția $\pi_{A1,A2}(r)$. Rezultatul operației este:

A1	***A2***
DN	1
DN	12
DJ	105
DJ	1

Selecția

Operația de selecție – notată cu $\sigma_P(r)$ – este definită prin $\sigma_P(r) := \{s | (s \in r), P(s)\}$. Aplicarea acestui operator va returna o submulțime a relației *r* care verifică expresia condițională *P*. Expresia *P* se supune regulilor calculului propozițional. Mai multe condiții pot fi conectate prin operatori logici ($\wedge$ şi, $\vee$ sau, $\neg$ nu)

Exemplu: Pentru relația *r* din exemplul anterior definim $\sigma_{A1=DN \wedge A2=1}(r)$

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	1	78	555
DN	1	13	762

Produsul cartezian

Considerând două relații r_1 şi r_2 se defineşte produsul cartezian $r_1 \times r_2$ ca fiind $r_1 \times r_2 := \{pq | p \in r_1, q \in r_2\}$. Această operație presupune că cele două relații sunt disjuncte ($r_1 \cap r_2 = \phi$). Dacă nu sunt disjuncte se aplică operația de redenumire.

Exemplu

Considerăm relația r_1 de forma:

A1	***A2***	***A3***	***A4***
DN	1	100	500
DN	3	7	333
DN	50	113	700

şi r_2 de forma:

B1	***B2***	***B3***
DN	1	E
DN	3	P
DN	50	S

produsul lor cartezian este:

A1	***A2***	***A3***	***A4***	***B1***	***B2***	***B3***
DN	1	100	500	DN	1	E
DN	1	100	500	DN	3	P
DN	1	100	500	DN	50	S
DN	3	7	500	DN	1	E
DN	3	7	333	DN	3	P
DN	3	7	700	DN	50	S
DN	50	113	500	DN	1	E
DN	50	113	333	DN	3	P
DN	50	113	700	DN	50	S

Uniunea

Operația de uniune ($r_1 \cup r_2$) se defineşte ca $r_1 \cup r_2 := \{r | (r \in r_1) \vee (r \in r_2)\}$. Relațiile trebuie să aibă acelaşi număr de atribute şi atributele să fie compatibile.

Exemplu: Se consideră o relație r_1 de forma:

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	12	6	200
DN	1	78	555
DN	1	13	762

şi relația r_2 de forma:

A1	***A2***	***A3***	***A4***
DJ	105	1	000
DJ	105	1	780
DJ	1	45	300

uniunea lor $r_1 \cup r_2$ are forma:

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	12	6	200
DN	1	78	555
DN	1	13	762
DJ	105	1	000
DJ	105	1	780
DJ	1	45	300

Diferența

Operația de diferență ($r_1 - r_2$) se defineşte ca $r_1 \cup r_2 := \{r | (r \in r_1) \wedge (r \notin r_2)\}$. Relațiile r_1 şi r_2 trebuie să aibă acelaşi număr de atribute şi atributele trebuie să fie compatibile.

Exemplu: Se consideră o relație r_1 de forma:

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	12	6	200
DN	1	78	555
DN	1	13	762

şi relația r_2 de forma:

A1	***A2***	***A3***	***A4***
DN	12	6	200
DN	1	13	762

diferența lor $r_1 - r_2$ are forma:

A1	***A2***	***A3***	***A4***
DN	1	100	345
DN	1	78	555

Pe lângă aceste operații se mai pot defini intersecția şi îmbinarea (join) care însă sunt redundante. De exemplu, intersecția poate fi definită ca: $P \cap Q = P - (P - Q)$. Chiar dacă nu sunt strict necesari pentru a face setul de operatori să fie complet, aceşti noi operatori, şi alții asemenea, pot face interogările mai simple.

5.2. LIMBAJUL DE BAZE DE DATE SQL

SQL (Structured Query Language = Limbaj de Interogare Structurat) este un limbaj ataşat unor sisteme de gestiune a bazelor datelor relaționale ce se constituie în momentul de față într-o majoritate a preferințelor.

Modelul relațional a apărut în 1970 şi este rodul activității lui E.F. Codd ca director de cercetări la Centrul IBM din San Jose. Pornind de la acest model au fost create mai multe limbaje între care *IBM Sequel* (evoluat ulterior în *Sequel/2*), *System/R*, *IBM DB2*.

Pe baza experienței acumulate s-a ajuns la SQL standardizat de ANSI în 1986. În 1989 apare o nouă versiune care este îmbunătățită în 1992 sub numele de SQL/92 sau SQL 2. În 1999 se încearcă reglementarea utilizării de obiecte în limbaj şi apare SQL/99 sau SQL3.

SQL este simultan:

- un limbaj de definire a datelor,
- un limbaj de manipulare a datelor şi
- un limbaj de control a datelor.

Ca limbaj de definire a datelor (în engleză DDL – Data Definition Language, în franceză Langage de Définition de Données) SQL poate stabili structura unei baze de date şi a tabelelor, o poate modifica sau şterge.

SQL permite interogarea datelor dintr-o tabelă sau mai multe tabele ale bazelor de date relaționale. Totodată permite adăugarea, modificarea sau ştergerea datelor din tabele. Se poate deci spune că SQL este un limbaj de manipulare a datelor (în engleză DML – Data Manipulation Language, în franceză Langage de Manipulation de Données).

SQL este şi un limbaj de control a datelor (în engleză DCL – Data Control Language, în franceză Langage de Protection d'Accès). El face posibilă definirea diferitelor nivele de acces, gruparea utilizatorilor şi stabilirea permisiunii de a crea, interoga şi actualiza datele, tabelele şi baza de date.

În descrierea limbajului SQL există câteva convenții pe care este bine să le cunoaştem. Astfel, elementele indicate cu majuscule sunt cuvinte cheie. Semnul „|" indică posibilitatea de alege unul dintre elementele listate. Între paranteze pătrate sunt elementele opționale. Parantezele nu apar efectiv în comandă.

Sintaxa comenzilor SQL nu este sensibilă la tipul caracterelor şi deci putem scrie cu majuscule sau minuscule fără ca aceasta să genereze o eroare.

Se pot introduce comentarii prin caracterul „%" (ce urmează după acest caracter până la sfârşitul rândului) sau prin delimitatorii „/*" şi „*/"

5.3. SQL – LIMBAJ DE MANIPULARE A DATELOR

Această caracteristică a SQL este cea mai căutată şi utilizată. Dacă definirea accesului şi privilegiilor în sistem este problema administratorului, dacă în definirea de noi tabele sunt implicate persoane specializate, în interogare şi vizualizare sunt interesați toți cei care vin în contact cu datele.

Comanda de bază care caracterizează SQL este SELECT. Cunoaşterea şi utilizarea sa pune la dispoziție un instrument simplu dar puternic de manipulare a datelor.

COMANDA SELECT

Sintaxa comenzii SELECT

SELECT este comanda de bază în manipularea datelor.

Această comandă se bazează pe algebra relațională şi are următoarea sintaxă fundamentală:

```
SELECT [ALL | DISTINCT] <lista numelor de coloane>|*
     FROM <lista tabelelor>
     [WHERE <expresie logică de condiționare>]
     [ORDER BY < lista numelor de coloane>]
     [GROUP BY < lista numelor de coloane>]
     [HAVING < expresie logică de condiționare >]
```

Opțiunea ALL permite vizualizarea tuturor liniilor din interogare şi este implicită spre deosebire de DISTINCT care elimină dublurile. ***Lista numelor de coloane*** este o înşiruire de coloane separate prin virgulă. Dacă se doreşte afişarea tuturor coloanelor şi nu există posibilitatea confuziei se poate utiliza semnul ***** care permite acest lucru.

Lista tabelelor este o înşiruire de nume de tabele din baza de date separate prin virgulă. ***Expresia logică de condiționare*** este o expresie constituită cu ajutorul operatorilor logici şi a comparatorilor matematici şi permite filtrarea acelor linii care îndeplinesc condiția.

Execuția comenzii are următorii paşi: se efectuează produsul cartezian al liniilor din tabele, se evaluează expresia logică de condiționare, se elimină toate elementele care nu îndeplinesc condiția şi din liniile rămase se returnează coloanele indicate în lista numelor de coloane.

Exemple simple de utilizare a comenzii SELECT

Presupunând că în baza de date a fost definită o tabelă numită PODURI, putem vizualiza toate liniile cu comanda:

```
SELECT * FROM Poduri
```

În cazul în care suntem interesați să urmărim doar numele drumului şi poziția kilometrică a podurilor, considerând că acestea sunt păstrate în coloanele NumeDrum şi PozKm, comanda va avea forma:

```
SELECT NumeDrum, PozKm FROM Poduri
```

De notat că această comandă returnează implicit toate înregistrările din tabelă, chiar şi dacă unele dintre ele au fost introduse eronat de mai multe ori.

Pentru a avea numai o singura dată valorile şi a elimina dublurile adăugăm clauza DISTINCT şi comanda ia forma:

```
SELECT DISTINCT NumeDrum, PozK FROM Poduri
```

Trebuie ca operaţia să fie privită atent şi precaut pentru că DISTINC returnează numai prima linie întâlnită dintre cele similare şi există posibilitatea ca tocmai această linie să fie inutilizabilă.

Operatori în expresia logică de condiționare

Pentru obţinerea numai acelor linii care îndeplinesc o anumită condiţie, în clauza WHERE vom utiliza o expresie ce întoarce o valoare logică. Dacă valoarea este FALS atunci linia este respinsă de la selecţie.

Expresia are forma generală a expresiilor logice şi conţine: operatori logici (AND, OR, NOT), comparatori pe şiruri ??? (IN, BETWEEN, LIKE), comparaţii aritmetice şi pe şiruri de caractere (=, !=, >, <, >=, <=, <>, !>, !<) şi operaţii aritmetice şi pe şiruri (+, -, *, /, &, |, ^, ~, %). Ordinea de precedenţă este cea obişnuită în expresiile logice şi aritmetice dar poate fi modificată prin utilizarea parantezelor.

Exemple de utilizare a comenzii SELECT cu clauza WHERE

În ultimul exemplu de mai sus presupunem că suntem interesaţi să aflăm drumul şi poziţia kilometrică a podurilor care aparţin de un district numit NEGRILA.

```
SELECT DISTINCT NumeDrum, PozKm
     FROM Poduri
     WHERE District = "NEGRILA"
```

Dacă suntem interesaţi să listăm toate podurile care au lungimea mai mare decât o lungime minimă atunci comanda va avea forma:

```
SELECT *
     FROM Poduri
     WHERE Lungime > 40.0
```

Analog, dacă dorim să combinăm condiţiile şi să evidenţiem toate podurile care aparţin de o listă de districte şi au lăţimea cuprinsă într-un anumit interval:

```
SELECT *
   FROM Poduri
   WHERE (District IN("NEGRILA","ALBILA,"CERNILA"))
            AND (Latime BETWEEN  6.0 AND 9.0)
```

Operatorul LIKE asigură compararea conţinutului unui şir de caractere prin utilizarea unor caractere generice (în engleză "wildcard character", în franceză «caractères jokers»). Aceste caractere ţin locul unei secvenţe de caractere (%), unui

singur caracter (_) sau pot indica un interval în care un anumit caracter poate lua valori ([-]).

Exemple de utilizare a caracterelor generice

”%grinzi%” indică orice şir de caractere care conține în interior cuvântul grinzi. Eventuala secvență de caractere înlocuită de % poate şi nulă deci şirul poate să şi înceapă cu literele indicate în clar.

”_ _ _ escu” indică un cuvânt de 7 litere care se încheie cu „escu”. Un astfel de cuvânt poate lua valoarea Ionescu, Popescu, Tomescu etc.

”[A-C]%” indică orice cuvânt care începe cu A, B sau C, inclusiv literele însele.

Ordonarea rezultatelor

Rezultatele interogării se afişează în conformitate cu poziția lor în tabele. Este totuşi necesar adesea să se stabilească o anumită ordine. Acest lucru se poate face utilizând clauza ORDER BY urmată de o listă de coloane separate prin virgulă, fiecare dintre ele putând fi urmate de cuvântul cheie ASC (ascendent) sau DESC (descendent). Implicit ordonarea se face în ordine ascendentă.

Exemplu

```
SELECT Prenume, Nume, Salariu
      FROM Personal
      ORDER BY Salariu DESC, Nume, Prenume
```

Din tabela cu personalul se selectează angajații în ordinea descrescătoare a salariului. În caz de egalitate înregistrările sunt aranjate în ordinea alfabetică a numelor şi prenumelor. ORDER BY ordonează după valori numerice, alfanumerice sau combinații.

Gruparea rezultatelor

Uneori este important ca operațiile şi afişarea rezultatelor să se facă pe grup de înregistrări. Aceste operații sunt permise de includerea în comandă a clauzei GROUP BY. Funcțiile utilizabile în astfel de situații dau o imagine statistică a grupurilor. Funcțiile agregate ce pot fi utilizate sunt: AVG() pentru calculul mediei, COUNT() care numără liniile din grup, MAX() şi MIN() care aleg maximul respectiv minimul selecției, SUM() care calculează suma şi STD() sau STDDEV() pentru calculul deviației standard (ultima funcție este o extensie a limbajului şi depinde de implementare).

Exemplu

```
SELECT Sectie, COUNT(*), AVG(Salariu),
           Max(Salariu), SUM(Realizări)
      FROM Personal
      ORDER BY Sectie
      GROUP BY Sectie
```

Se grupează înregistrările angajaţilor din fiecare secţie şi se afişează numele secţiei, numărul angajaţilor, media şi maximul salarial precum şi suma realizărilor.

Lucrul cu joncţiuni

Exemplele de interogare de mai sus au fost date pentru lucrul cu o singură tabelă. Totuşi sintaxa clauzei FROM este:

```
FROM <lista tabelelor>
```

permiţând includerea numelui mai multor tabele simultan.

Pentru a simplifica structura tabelelor şi a minimiza spaţiul utilizat, datele trebuie stocate convenabil în mai multe tabele. În momentul utilizării datele din tabele trebuie combinate. Acest lucru este posibil prin utilizarea joncţiunilor.

Joncţiunea se realizează prin efectuarea produsului cartezian al tabelelor implicate în ordinea precizată de clauza ORDER BY. Clauza WHERE elimină toate elementele produsului cartezian care nu îndeplinesc expresia logică de condiţionare. Deoarece în două tabele diferite pot apare coloane cu acelaşi nume se identifică unic fiecare coloană prin indicarea numelui tabelei urmat de caracterul ”.” urmat de numele coloanei.

Acest tip de joncţiune este opţiunea naturală şi se numeşte joncţiune internă (inner join).

Dacă alegerea prin clauza WHERE se face prin verificarea egalităţii a două coloane atunci joncţiunea poartă numele de echi-joncţiune (joncţiune echivalentă).

Exemplu

```
SELECT NumeDrum, PozKm, NumeDistr
      FROM Poduri, District
      WHERE Poduri.CodDistr = District.CodDistr
```

Dacă însă alegerea prin clauza WHERE se face prin verificarea unei inegalităţii între coloane sau combinaţii de coloane atunci joncţiunea poartă numele de non-echi-joncţiune (joncţiune neechivalentă).

Joncțiuni externe

O joncțiune externă este o joncțiune în care fiecare înregistrare corespunzătoare din cele două tabele este inclusă într-o înregistrare rezultat chiar dacă nu îndeplineşte condițiile. Asemenea înregistrare va furniza coloane cu conținut NULL.

Joncțiunea externă stânga

Joncțiunea externă stânga sau simplu joncțiunea stânga include liniile din prima tabelă a unei joncțiuni (cea din stânga) chiar dacă nu satisfac condiția impusă.

Exemplu:

```
SELECT INSPECT.*, DEFECT.*
FROM INSPECT LEFT JOIN DEFECT
     ON INSPECT.ID = DEFECT.ID_INSPECT;
```

Joncțiunea externă dreapta

Joncțiunea externă dreapta sau joncțiunea dreapta include liniile din a doua tabelă a unei joncțiuni (cea din dreapta) chiar dacă nu satisfac condiția impusă.

Exemplu:

```
SELECT INSPECT.*, DEFECT.*
FROM INSPECT RIGHT JOIN DEFECT
     ON INSPECT.ID = DEFECT.ID_INSPECT;
```

Utilizarea interogărilor imbricate

În cadrul unei clauze se poate utiliza o singură valoare. Uneori este nevoie a se utiliza mai multe valori care să fie preluate din tabel. Pentru generarea acestor valori se poate utiliza o nouă interogare. Interogările imbricate se numesc sub-interogări. Acest mod de lucru se mai numeşte interogare în cascadă deoarece se aşteaptă încheierea unei interogări pentru a începe o alta.

Sub-interogările trebuie plasate într-o clauză WHERE sau HAVING unde înlocuieşte o constantă sau un grup de constante care intră într-o relație.

Când înlocuieşte o constantă sub-interogarea trebuie să întoarcă o singură valoare (o tabelă cu o singură linie şi o singură coloană):

```
SELECT ---
     FROM ---
     WHERE --- < (
          SELECT --- FROM ---
          )
```

Când înlocuieşte un grup de constante incluse într-o expresie care conține operatorii IN, EXISTS, ALL, ANY sub-interogarea trebuie să întoarcă un set de valori cu o singură dimensiune:

```
SELECT ---
     FROM ---
     WHERE --- IN (
          SELECT --- FROM ---
          )
```

Interogările imbricate respectă toate regulile de sintaxă şi de logică ale comenzii SELECT şi pot avea la rândul lor sub-interogări..

Exemplu

Presupunând că în tabela Poduri avem coloana IndStare de tip numeric în care se stochează un indicator de stare tehnică. Cu următoarea comandă vom afişa toate podurile la care indicele de stare este sub medie.

```
SELECT *
     FROM Poduri
     WHERE IndStare <
          (
          SELECT AVG(IndStare) FROM Poduri
          )
```

OPERAȚII DE ASAMBLARE

Rezultatul comenzilor de interogare poate fi combinat utilizând operațiile definite de algebra relațională. Aceste operații se efectuează între două comenzi SELECT. Operatorii sunt UNION, INTERSECT şi EXCEPT.

Operatorul UNION

Operatorul UNION realizează reuniunea setului de tuple generate ca rezultat a două comenzi SELECT. Sintaxa are forma:

```
SELECT ---
UNION [ALL]
SELECT ---
```

Comenzile SELECT respectă sintaxa şi regulile generale prezentate anterior. Implicit dublurile se elimină. Pentru a afişa toate liniile se va utiliza clauza UNION ALL.

Exemplu

Pentru a prezenta o listă completă a adreselor complete a tuturor unităților din administrație. Datele sunt păstrate în două tabele Directii şi Sectii. Pe fiecare dintre cele două tabele se execută o interogare care are ca rezultat coloanele Denumire, Adresa, Telefon. Rezultatele sunt reunite prin operatorul UNION. Comanda completă este:

```
SELECT Denumire, Adresa, Telefon
     FROM Sectii
UNION
SELECT Denumire, Adresa, Telefon
     FROM Districte
```

Operatorul INTERSECT

Operatorul INTERSECT realizează operația de intersectare a seturilor de tuple generate ca rezultat a două comenzi SELECT. Sintaxa are forma:

```
SELECT ---

INTERSECT

SELECT ---
```

Comenzile SELECT respectă sintaxa şi regulile generale prezentate anterior. Ambele trebuie să aibă aceeaşi schemă. Nu este o comandă standard şi nu este implementată de toate SGBDR. Nu este un operator necesar. Poate fi înlocuit prin comenzi SELECT imbricate, de exemplu operația:

```
SELECT a1,b1 FROM Tabela1
INTERSECT
SELECT a2,b2 FROM Tabela2
```

poate fi înlocuită prin:

```
SELECT a1,b1 FROM Tabela1
     WHERE EXISTS (  SELECT a2,b2 FROM Tabela2
                     WHERE a1=a2 AND b1=b2 )
```

Operatorul EXCEPT

Operatorul EXCEPT realizează operația de diferență a seturilor de tuple generate ca rezultat a două comenzi SELECT, adică se păstrează numai acele linii care fac parte din primul set dar nu fac parte din cel de-al doilea.

Sintaxa are forma:

```
SELECT ---
INTERSECT
SELECT ---
```

Comenzile SELECT respectă sintaxa şi regulile generale prezentate anterior. Ambele trebuie să aibă aceeaşi schemă. Nu este o comandă standard şi nu este implementată de toate SGBDR. Nu este un operator absolut necesar. Poate fi înlocuit prin comenzi SELECT imbricate, de exemplu operaţia:

```
SELECT a1,b1 FROM Tabela1
EXCEPT
SELECT a2,b2 FROM Tabela2
```

poate fi înlocuită prin:

```
SELECT a1,b1 FROM Tabela1
      WHERE NOT EXISTS (SELECT a2,b2 FROM Tabela2
                              WHERE a1=a2 AND b1=b2 )
```

ADĂUGAREA DE DATE ÎNTR-O TABELĂ

SQL permite utilizarea comenzii INSERT pentru adăugarea de noi linii într-o tabelă fie direct prin impunerea valorilor de către utilizator (clauza VALUES), fie prin preluarea lor dintr-o interogare (clauza SELECT).

Adăugarea de linii prin valoare

Introducerea de noi linii se face prin comanda INSERT şi specificarea tabelei destinaţie prin clauza INTO la care se specifică valorile efective prin clauza VALUES astfel:

```
INSERT INTO NumeTabela
             (coloana1,coloana2,coloana3,...)
      VALUES (valoare1,valoare2,valoare3,... )
```

Se adaugă o singură linie. Valorile trecute în clauza VALUES sunt atribuite fiecărei coloane în ordinea enumerării. Coloanele care nu sunt cuprinse în listă primesc valoarea NULL.

Dacă o coloană nu poate rămâne nulă atunci se generează o eroare. Valorile trebuie să fie compatibile cu tipul declarat al coloanei.

Dacă într-o coloană se acceptă doar valoare unică (clauza UNIQUE la creare) şi se adaugă o valoare deja existentă se generează o eroare.

Adăugarea de linii obținute dintr-o interogare

Introducerea de noi linii se face prin comanda INSERT şi specificarea tabelei destinație prin clauza INTO la care se adaugă o clauza SELECT:

```
INSERT INTO NumeTabela
            (coloana1,coloana2,coloana3,...)
      SELECT Col1, Col2, Col3, ...
      FROM NumeTabel2
      WHERE <expresie logică de condiționare>
```

Se adaugă numai una, mai multe sau nici o linie funcție de îndeplinirea condiției ataşate clauzei WHERE.

Dacă într-o coloană se acceptă doar valoare unică (clauza UNIQUE la creare) şi se adaugă o valoare deja existentă se generează o eroare.

Se respectă aceleaşi reguli ca la adăugarea de linii prin valoare. Atribuirea valorilor se face în ordinea în care au fost incluse în listă. Nu este obligatoriu ca o coloană din lista clauzei SELECT să aibă acelaşi nume cu coloana corespunzătoare din lista clauzei INTO. În lista de coloane inclusă în clauza SELECT nu trebuie să apară coloane din tabela NumeTabela.

Exemple

1. Adăugarea unei linii prin declararea explicită a valoarilor:

```
INSERT
      INTO Poduri(    NumeDrum, PozKm, Obstacol,
                      Lungime, Latime, District)
      VALUES (   "DN87Q", 89.779, "Râul Repedea",
                 167.78, 12.40, "NEGRILA" )
```

2. Adăugarea unei linii cu valori obținute dintr-o interogare

```
INSERT
      INTO ManvP( Drum, Km, Obstacol,
                  LungPOD, LatPod, District)
      SELECT NumeDrum, PozKm, Obstacol,
             Lungime, Latime, District
      FROM Poduri
      WHERE District IN("NEGRILA","ALBILA,"CERNILA")
           AND  (Lungime > 10.0)
```

MODIFICAREA DATELOR DINTR-O TABELĂ

Datele deja introduse într-o tabelă pot fi modificate atunci când condiţiile o cer. Pentru aceasta standardul SQL a definit comanda UPDATE. Aceasta are sintaxa:

```
UPDATE NumeTabela
      SET   Coloana1 = valoare1
            [, Coloana2 = valoare2 ]
            [, Coloana3 = valoare3 ]
            [ ... ]
      WHERE <expresie logică de condiţionare>
```

Operaţia de modificare se efectuează în tabela NumeTabela, fiecăreia dintre coloanele specificate în clauza SET i se atribuie valoarea de după semnul "=". Valoarea ce se atribuie poate să provină şi din evaluarea unei expresii deci fiecare linie poate primi un set deosebit de valori pentru coloanele specificate. Atribuirea se aplică numai acelor linii care îndeplinesc expresia logică de condiţionare din clauza WHERE.

Exemplu

```
UPDATE Poduri
      SET District = "FRUMUSENI"
      WHERE (District ="NEGRILA") and
            (NumeDrum = "DN87Q") and
            (PokKm<= 109.543)
```

Când un anumit sector de drum a fost transferat între două districte şi simultan lucrările de artă aferente trec din administrarea unui district în a celuilalt se poate actualiza conţinutul coloanei District în tabela Poduri.

ELIMINAREA DATELOR DINTR-O TABELĂ

Când anumite date înregistrate în liniile tabelelor devin inutile ele pot fi şterse pentru a nu încărca inutil spaţiul rezervat bazei de date şi pentru a nu complica comenzile de interogare.

Pentru aceasta SQL oferă comanda DELETE cu următoarea sintaxă:

```
DELETE FROM NumeTabela
      WHERE <expresie logică de condiţionare>
```

Comanda DELETE are clauzele FROM pentru a indica numele tabelei în care se execută operaţia şi WHERE pentru a face selecţia liniilor care se elimină.

Exemplu

După ce un sector de drum a fost trecut dintr-o categorie în alta, lucrările de artă aferente nu se mai află în administrarea instituției care posedă baza de date. Din considerente istorice se pot păstra datele în arhive, ceea ce presupune un transfer utilizând comanda INSERT, dar datele despre fiecare în parte se şterg din tabela de gestiune curentă. Pentru aceasta utilizăm o comandă DELETE de forma:

```
DELETE FROM Poduri
     WHERE (NumeDrum="DN87Q")and(PokKm<=109.855)
```

FUNCȚII UTILIZATE ÎN INTEROGĂRI

În realizarea interogărilor se utilizează, pe lângă funcțiile agregate, şi a operatorilor aritmetici un număr de funcții matematice care asigură prelucrarea corectă a datelor. Aceste funcții nu sunt standard şi diferă de la implementare la implementare. Totuşi câteva se regăsesc în majoritatea variantelor SQL. Prezentăm mai jos aceste funcții cu precizarea că înainte de a le folosi trebuie să verificăm în documentația aflată la dispoziție prezența şi sintaxa lor exactă.

Tabelul 2 Funcții matematice utilizate în SQL

Funcția	**Descriere**
ABS(X)	Valoarea absolută a numărului real X
CEIL(X)	X is a decimal value that will be rounded up.
FLOOR(X)	X is a decimal value that will be rounded down.
GREATEST(X,Y)	Valoarea ce mai mare dintre X şi Y
LEAST(X,Y)	Valoarea cea mai mică dintre X şi Y
MOD(X,Y)	Restul împărțirii întregi a valorilor X şi Y (X modulo Y)
POWER(X,Y)	Returnează X la puterea Y
ROUND(X,Y)	Rotunjeşte X cu Y zecimale. Dacă Y este omis, X este rotunjit la cel mai apropiat întreg
SIGN(X)	Semnul valorii X
SQRT(X)	Rădăcina pătrată a valorii X

Pe lângă acestea se utilizează şi funcții de procesare a şirurilor de caractere. Similar celor matematice aceste funcții nu sunt standard şi trebuie verificată sintaxa lor exactă.

Tabelul 3 Funcţii de manipularea şirurilor de caractere

LEFT(*<şir>*,X)	Returnează X caractere din stânga şirului de caractere
RIGHT(*<şir>*,X)	Returnează X caractere din dreapta şirului de caractere
UPPER(*<şir>*)	Converteşte caracterele şirului în majuscule
LOWER(*<şir>*)	Converteşte caracterele şirului în minuscule
INITCAP(*<şir>*)	Converteşte începutul propoziţiilor în majuscule
LENGTH(*<şir>*)	Returnează lungimea şirului de caractere
<şir1>||*<şir2>*	Concatenează conţinutul celor două şiruri în ordinea aleasă
LPAD(*<şir>*,X,'*')	Adaugă la stânga şirului numărul necesar de caractere cuprinse între apostrof pentru a ajunge la lungimea X
RPAD(*<şir>*,X,'*')	Adaugă la dreapta şirului numărul necesar de caractere cuprinse între apostrof pentru a ajunge la lungimea X
SUBSTR(*<şir>*,X,Y)	Extrage din şir Y litere începând cu poziţia Y
NVL(*<col>*,*<val>*)	Pentru valorile NULL ale coloanei *col* returnează valoarea *val*, altfel returnează valoarea coloanei

În afara acestor funcţii există şi altele specifice fiecărei implementări. Pentru lista completă şi sintaxa corectă trebuie studiată documentaţia care se livrează la cumpărarea pachetelor software.

5.4. SQL – LIMBAJ DE DEFINIRE A DATELOR

CREAREA TABELELOR

SQL permite definirea de tabele noi. Acest lucru asigură flexibilitate în procesare şi posibilitatea de dezvoltare a aplicaţiilor prin lărgirea tipului şi cantităţii de date stocate.

Comanda specifică pentru adăugarea de tabele în baza de date este „CREATE TABLE". Prin această comandă se specifică numele tabelei care se creează, numele, tipul şi caracteristicile coloanelor care i se ataşează.

Sintaxa comenzii CREATE TABLE

```
CREATE TABLE NumeTab
    (NumeCol1 TipDeDate [NOT NULL][UNIQUE | INDEX],
     NumeCol2 TipDeDate [NOT NULL][UNIQUE | INDEX],
     NumeCol3 TipDeDate [NOT NULL][UNIQUE | INDEX],
      ... )
```

Trebuie să existe cel puţin un nume de coloană în listă. Numele de coloană trebuie să respecte regulile generale din SQL care seamănă cu regulile din alte

limbaje. Un nume începe obligatoriu cu o literă. Într-o tabelă pot exista maximum 254 de coloane. Fiecare coloană specificată are un anumit tip de date. Opțiunea NOT NULL, situată după fiecare declarare de tip, precizează obligativitatea atribuirii unei valori la inserare, editare, modificare pentru coloana corespunzătoare.

În zona de opțiuni a descrierii coloanei poate apare şi clauza DEFAULT care este prezentată mai jos.

Crearea de tabele cu copierea datelor

Este posibil ca la crearea unui tabele să se adauge date ce pot fi preluate din alte tabele. Sintaxa comenzii de creare este:

```
CREATE TABLE
      NumeTabel (Coloana1 TipColoana1,
                 Coloana2 TipColoana2,
                 Coloana3 TipColoana3,
                       ...
                 )
      AS SELECT NumeCâmp1,NumeCâmp2,NumeCâmp3,...
      FROM NumeTabel2
      [WHERE <expresie logică de condiționare>]
```

În comandă trebuie să fie inclusă cel puțin o definiție de coloană şi, corespunzător, o definiție de câmp în clauza AS SELECT.

Tipuri de date în tabelele bazelor de date

Tipul de date declarat pentru fiecare coloană trebuie să fie din lista de tipuri acceptate de fiecare implementare în parte. Conform ANSI/ISO tipurile de date pentru coloane sunt prezentate în tabelul următor.

Tabelul 4. Tipuri de date definite de ANSI / ISO

Tip	**Descriere**
CHAR[(n)]	Şir de caractere cu lungime fixă
VARCHAR[(n)]	Şir de caractere cu lungime variabilă
NUMBER[(n,[d])]	Număr pe de n cifre [opțional d cifre după virgulă]
SMALLINT	Întreg pe 16 biți (-32768 la 32757)
INTEGER	Întreg pe 32 de biți (-2E31 la 2E31-1)
FLOAT[(n)]	Număr în virgulă mobilă
DOUBLE PRECISION	Număr real dublă precizie
DATE	Date de tipul dată

Pe lângă acestea există multe alte tipuri care au fost definite numai în câteva implementări. Unele dintre cele mai întâlnite sunt TIME care stochează ora în formatul 12:59:59.99 şi TIMESTAMP care stochează data şi ora.

Respectând în linii mari prescripţiile ANSI/ISO fiecare implementare a încercat să satisfacă nevoilor specifice cărora trebuie să le răspundă.

Ca un exemplu Oracle7 suportă următoare tipuri:

Tabelul 5. Tipuri de date suportate de Personal Oracle7

Tip	**Descriere**
CHAR	Şiruri de caractere cu lungimea între 1 şi 255. La şirurile mai scurte se adaugă spaţii la dreapta.
DATE	Include anul, luna, ziua, ora minutul şi secunda.
LONG	Şir de caractere cu lungime variabilă de până la 2Gigaocteţi.
LONG RAW	Date binare de până la 2 Gigaocteţi.
NUMBER	Date numerice.
RAW	Date binare de până la 255 de octeţi.
ROWID	Valoare hexazecimală reprezentând o adresă unică a liniei.
VARCHAR2	Şir de lungime variabilă până la 2000 de caractere.

Note:

1. Tipul LONG mai este numit şi MEMO în alte sisteme de gestiune. Se foloseşte pentru stocarea de texte foarte lungi.

2. Tipul LONG RAW mai este numit şi BLOB (Binary Large OBject). Poate servi pentru stocarea de date reprezentate binar precum fotografii, grafice, sunet, imagini în mişcare etc. Acest tip de date este cel mai des utilizat în sistemele multimedia.

Tipurile de date şi structura lor evoluează. Ele devin tot mai complexe pe măsură ce noi concepte apar în informatică. În setul standard se includ tipuri de date mulţime, tablouri de date, date de tip obiecte grafice, posibilitatea ca utilizatorul să-şi definească propriile structuri şi tipuri de date care să poată fi utilizate ulterior pentru stocare.

Un exemplu este dat de evoluţia tipurilor standard de date definite pentru SQL99 comparativ cu tipurile de date definite în standardele anterioare.

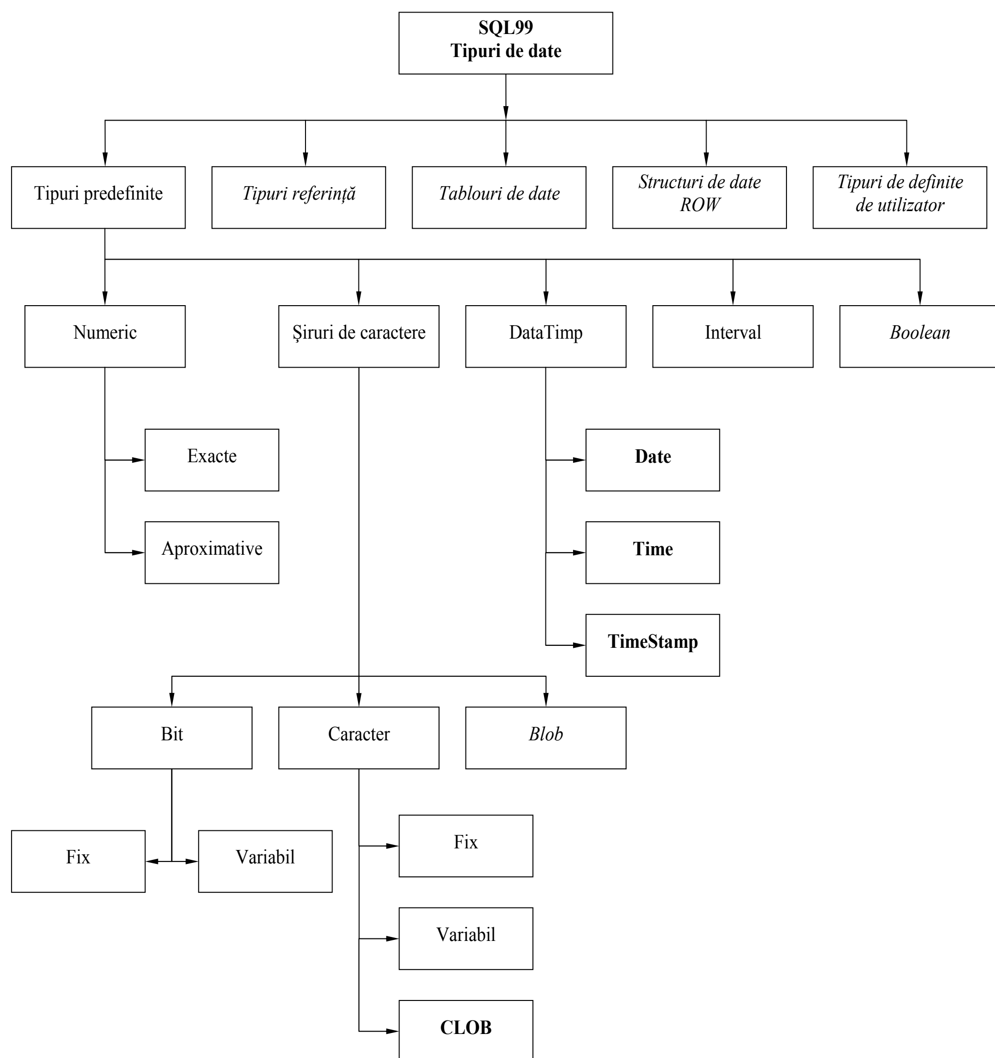


Fig. 33 Tipuri de date în SQL 1999

Exemple de tipuri de date implementate în diferite versiuni de SQL sunt numeroase dar ne-am limitat la cele mai cunoscute. Pentru implementarea exactă se vor studia manualele de referință sau documentația furnizată de producător ce însoțeşte pachetele de instalare la cumpărare.

Expresii de restricție

Expresiile de restricție sunt clauze ce forțează valorile introduse de utilizator într-o coloană să respecte anumite reguli.

Aceste reguli se definesc la crearea tabelei cu ajutorul unor cuvinte cheie listate în continuare:

- DEFAULT
- NOT NULL
- UNIQUE
- CHECK

Regulile introduse prin aceste clauze au efect pe toată durata existenței tabelei sau până când tabela este alterată.

Definirea valorii implicite

Dacă pentru unul dintre câmpurile dintr-o înregistrare nu se introduce nici o valoare, SQL permite asocierea unei valori implicite. Aceasta se realizează cu clauza DEFAULT. Cuvântul cheie DEFAULT trebuie să fie urmat de:

- O constantă numerică;
- O valoare şir de caractere;
- Cuvântul cheie USER (numele utilizatorului);
- Cuvântul cheie NULL;
- Cuvântul cheie CURRENT_DATE (data introducerii);
- Cuvântul cheie CURRENT_TIME (ora introducerii);
- Cuvântul cheie CURRENT_TIMESTAMP (data şi ora introducerii);

La crearea unei noi înregistrări se asignează automat valoare ataşată clauzei DEFAULT. Prin aceasta ne putem asigura că un câmp nu este niciodată gol.

Impunerea introducerii unei valori

Clauza NOT NULL impune ca la editarea unui câmp să se introducă obligatoriu o valoare pentru un câmp. Dacă acest câmp este gol atunci sistemul nu permite salvarea înregistrării.

Verificarea unicității

Clauza UNIQUE permite verificarea faptului că valoarea unui câmp nu există deja în tabelă. Aceasta garantează că toate valorile din câmp sunt diferite.

Verificarea unei condiții

SQL permite ca la introducerea unei valori într-un câmp să se facă o verificare a unei condiții logice. Aceasta se va face cu clauza CHECK(<*expresie logică*>). Sistemul evaluează expresia logică şi dacă aceasta nu se verifică atunci valoarea nu se acceptă.

Expresia logică are acelaşi format ca expresia ataşată clauzei WHERE din comanda SELECT.

Exemplu de utilizarea a restricţiilor

Prezentăm în continuare un exemplu de creare a unei tabele prin comanda CREATE TABLE în care se utilizează clauzele de restricţie.

```
CREATE TABLE Pod(
      ID      char(9) NOT NULL UNIQUE
      TipDrum char(2) NOT NULL DEFAULT "DN",
      Drum    char(4) NOT NULL,
      Km      integer(3) CHECK(Km>0),
      M       integer(3) CHECK(M>0)
      AnContr integer check (AnConstr > 1850)
      )
```

CREAREA UNEI VIZIUNI

În viaţa reală, fiecare poate avea propria imagine despre elementele lumii înconjurătoare. Această imagine poate fi descrisă în conformitate cu propriile concepte, din perspectivă proprie. Această reprezentare proprie poartă numele de viziune ("view" în engleză, « vue » în franceză; în limba română s-a mai utilizat termenul de „vedere" şi de „relaţie virtuală")

În SQL o viziune este o tabelă virtuală. Datele pot proveni din mai multe tabele şi sunt grupate în conformitate cu reprezentarea dorită de utilizator. Fiecare utilizator al sistemului SQL îşi poate crea o viziune proprie asupra datelor din baza de date. Datele din diferite tabele pot fi combinate după dorinţă într-o unitate logică astfel încât în continuare ele să apară ca un tot unitar, virtual într-o singură tabelă.

Această operaţie se realizează utilizând comanda CREATE VIEW a cărei sintaxă este următoarea:

```
CREATE VIEW NumeViziune(Coloana1,Coloana2,Coloana3,...)
      AS SELECT ...
```

Clauza AS SELECT este urmată de toate opţiunile normale ale comenzii SELECT (vezi). Este, în fapt, declararea formală a unei comenzi SELECT şi ataşarea unui nume, astfel încât ori de câte ori dorim să repetăm acea comandă să nu mai fie nevoie să scriem toate liniile, clauzele şi opţiunile.

O formă uzuală, dar nu singura, poate fi:

```
CREATE VIEW Viziune1 (Coloana1,Coloana2,Coloana3)
      AS SELECT Alias1.Col1,Alias1.Col2,Alias2.Col3
            FROM Tabela1 Alias1, Tabela2 Alias2
            WHERE (Alias1.Col2=Alias2.ColX)
```

În continuare rezultatul interogării va fi privit ca o tabelă care poate fi, la rândul ei, utilizată în comenzi de interogare.

Utilitatea viziunilor rezidă în faptul că acestea se constituie într-un fel de interfaţă între baza de date şi utilizator.

Utilizarea viziunilor are avantaje multiple, între care:

- Afişează o selecţie convenabilă din baza de date,
- Restricţionează accesul altor utilizatori, asigurând o securitate sporită,
- Grupează informaţiile care sunt privite ca o singură entitate, chiar dacă ele provin din tabele diferite.

CREAREA ŞI UTILIZAREA INDECŞILOR

Un index este un auxiliar care facilitează accesul la baza de date în condiţiile sortării elementelor. Un index nu este absolut necesar, există mecanisme de sortare şi acces independente, dar ajută la economisirea timpului de execuţie a comenzilor de căutare, selecţie şi ordonare. Totuşi, reversul este prelungirea unora dintre comenzile de adăugare şi modificare de date. De aceea este necesar ca utilizarea unui index să fie bine gândită şi justificată prin balanţa avantajelor şi dezavantajelor. Este necesar a se evita în baza de date dublurile care sunt, în SQL, destul de complicat de tratat. Unele sisteme rezolvă această problemă prin crearea automată a unui câmp suplimentar cu valoare unică asupra căruia se defineşte o cheie primară.

Crearea unui index se face în SQL prin comanda CREATE INDEX. Aceasta are sintaxa următoare:

```
CREATE [UNIQUE] INDEX NumeIndex
      ON NumeTabela
      (Camp1 [ASC/DESC],  ... )
```

Opţiunea UNIQUE permite eliminarea dublurilor. Absenţa acestei clauze permite prezenţa tuturor liniilor. Opţiunea ASC/DESC defineşte ordinea ascendentă sau descendentă a valorilor în coloane. Implicit ordinea este ascendentă.

MODIFICAREA TABELELOR ŞI A BAZEI DE DATE

Într-o bază de date este posibil să se elimine sau să se modifice tabelele, indecşii şi viziunile. Pentru tabelele deja create se poate modifica structura sau se poate schimba numele. Se pot adăuga coloane, şterge coloane sau se poate modifica tipul datelor stocate într-o coloană.

Ştergerea componentelor bazei de date

Se poate renunţa la prezenţa unor tabele, indecşi şi viziuni atunci când acestea devin inutile. Pentru aceasta se utilizează comanda DROP. Această comandă trebuie bine gândită înainte de a fi utilizată pentru că efectele sale sunt definitive şi ireversibile.

Sintaxa pentru ştergerea unui tabele este:

```
DROP TABLE NumeTabela
```

Sintaxa pentru ştergerea unui index este:

```
DROP INDEX NumeIndex
```

Sintaxa pentru ştergerea unei viziuni este:

```
DROP VIEW NumeViziune
```

Ştergerea datelor dintr-o tabelă

Comanda DROP TABLE presupune ştergerea datelor şi eliminarea tabelei în totalitate. Dacă, însă, dorim să păstrăm tabela cu structura anterioară şi să eliminăm numai datele care nu ne mai sunt necesare atunci putem să utilizăm comanda TRUNCATE. Sintaxa comenzii este:

```
TRUNCATE TABLE NumeTabela
```

Redenumirea unei tabele

Dacă se doreşte păstrarea datelor şi a tabelei dar schimbarea numelui tabelei se poate utiliza comanda RENAME. Sintaxa comenzii este:

```
RENAME NumeVechi TO NumeNou
```

Unele sisteme de gestiune a datelor nu implementează această comandă. Este necesar a se face o documentare adecvată înainte de utilizarea comenzii.

Eliminarea unor coloane dintr-o tabelă

Dacă se doreşte modificarea structurii unei tabele în sensul ştergerii unei coloane se va utiliza comanda ALTER cu clauza DROP COLUMN. Sintaxa comenzii este:

```
ALTER TABLE NumeTabela
      DROP COLUMN NumeColoana
```

Înainte ca să executăm această comandă trebuie să verificăm dacă respectiva coloană face parte dintr-o viziune, dintr-un index, sau dacă este obiectul unei verificări de integritate. Când una dintre aceste stări este îndeplinită comanda de ştergere nu se execută şi se generează o eroare.

Adăugarea unei coloane într-o tabelă

Se poate adăuga o coloană nouă într-o tabelă utilizând comanda ALTER TABEL cu clauza ADD. Sintaxa comenzii este:

```
ALTER TABLE NumeTabela
      ADD NumeColoana TipDate
```

Modificarea unei coloane

Modificarea declaraţiei unei coloane dintr-o tabelă se face utilizând comanda ALTER TABEL cu clauza MODIFY.

Sintaxa comenzii ALTER TABLE este:

```
ALTER TABLE NumeTabela
      MODIFY NumeColoana TipDate
```

Unele implementări au înlocuit clauza MODIFY cu clauza ALTER COLUMN restul comenzii rămânând identic.

5.5. SQL – LIMBAJ DE CONTROL A DATELOR

Mai mulţi utilizatori pot accesa simultan o bază de date. Fiecare dintre aceştia pot avea necesităţi diferite. Unii pot şi trebuie să modifice datele, alţii trebuie numai să le citească. Pe baza necesităţilor se pot defini drepturile acordate fiecărei persoane care accesează baza de date.

Standardul SQL permite definirea drepturilor de acces cu ajutorul clauzelor:

- GRANT pentru acordarea de drepturi, şi
- REVOKE pentru retragerea drepturilor.

Aceste drepturi denumite şi privilegii sunt:

- DELETE – pentru ştergerea de date;
- INSERT – pentru adăugarea de noi înregistrări;
- UPDATE – pentru actualizarea datelor.

- SELECT – pentru citirea datelor;

Utilizatorul care a creat un tabel, viziune sau index poate acorda sau retrage privilegiile asupra entității în cauză. Aceste drepturi pot fi retransmise dacă se specifică astfel.

ATRIBUIREA PRIVILEGIILOR

Prin comanda GRANT se pot acorda privilegii unuia sau mai multor utilizatori asupra unui sau mai multor elemente din baza de date.

Comanda GRANT are formatul:

```
GRANT <listă  de  drepturi>
      ON <listă  de  componente>
      TO <listă de utilizatori>
      [WITH GRANT OPTION]
```

Lista de drepturi poate fi înlocuită cu cuvântul cheie ALL şi se acordă toate privilegiile. Privilegiile acordate pentru o tabelă pot fi limitate la un număr de coloane prin asocierea listei de coloane la lista de drepturi. În locul listei de utilizatori se poate utiliza cuvântul cheie PUBLIC şi privilegiile acordate se transmit tuturor utilizatorilor.

Clauza WITH GRANT OPTION autorizează utilizatorii să transmită mai departe privilegiile primite prin comanda în curs.

Exemplu:

```
GRANT UPDATE(Defecte, Comentarii)
      ON Inspect
      TO InsPod03
      WITH GRANT OPTION
```

RETRAGEREA PRIVILEGIILOR

Dacă drepturile acordate nu mai sunt necesare, ele pot fi retrase. Comanda REVOKE asigură retragere drepturilor unui utilizator.

```
REVOKE
      [GRANT OPTION FOR] <listă  de  drepturi>
      ON <listă  de  componente>
      FROM <listă de utilizatori>
```

Clauza WITH GRANT OPTION retrage utilizatorilor dreptul de a transmite mai departe privilegiile primite prin comanda GRANT.

Lista de drepturi poate fi înlocuită cu cuvântul cheie ALL şi se retrag toate privilegiile. Privilegiile retrase pentru o tabelă pot fi limitate la un număr de coloane

prin asocierea listei de coloane la lista de drepturi. În locul listei de utilizatori se poate utiliza cuvântul cheie PUBLIC şi privilegiile se retrag tuturor utilizatorilor.

CONTROLUL TRANZACŢIILOR

Sistemele de gestiune a bazelor de date trebuie să ofere mecanisme de asigurarea a consistenţei datelor.

De exemplu, dacă se şterge o înregistrare dintr-o tabelă de inspecţii trebuie să se şteargă şi înregistrările corespunzătoare dintr-o tabelă cu defectele constatate în timpul inspecţiei. Totodată, dacă se introduce o valoare de apreciere pentru un defect atunci trebuie să se reevalueze starea tehnică. Toate acestea şi multe altele se fac automat prin controlul tranzacţiilor.

Un astfel de mecanism de control al tranzacţiilor se numeşte în SQL trigger (declanşator).

Trigerii sunt funcţii a căror acţiune este iniţiată de evenimente care apar în sistem. Se pot urmări patru categorii de evenimente:

- Crearea unei noi înregistrări;
- Ştergerea unei înregistrări existente;
- Actualizarea unei înregistrări existente;
- Citirea unei înregistrări existente.

Ultimul dintre evenimente este mai rar utilizat şi de aceea uneori nici nu este implementat.

Un trigger se poate crea cu comanda CREATE TRIGGER care are sintaxa următoare:

```
CREATE TRIGGER <nume>
     {BEFORE | AFTER} {<eveniment>[OR ...]}
     ON <tabela> FOR EACH { ROW | STATEMENT }
     EXECUTE PROCEDURE <funcţie ( argumente )>
```

unde elementele opţionale din sintaxă semnifică:

nume reprezintă numele sub care trigger-ul va fi recunoscut. Numele trebuie să fie distinct.

eveniment poate fi una dintre opţiuni: INSERT, DELETE, UPDATE.

tabela numele tabelei pentru care se aplică trigger-ul.

funcţia numele funcţiei care va execuatată.

argumente lista de argumente care se furnizează funcției.

Acțiunea unui trigger declanşează înainte (BEFORE) sau după (AFTER) o încercare de modificare unei înregistrări (ROW) sau a întregii tabele (STATEMENT). Dacă operația se declanşează înainte atunci prin funcția apelată se poate renunța la desfăşurarea evenimentului, dacă trigger-ul este inițiat după eveniment atunci toate modificările sunt vizibile.

Mai multe triggere ataşate aceluiaşi eveniment al unei tabele tratarea lor se inițializează în ordine alfabetică.

Se poate renunța la utilizarea serviciului unui trigger prin eliminarea acestuia din sistem. Un trigger este eliminat prin comanda DROP TRIGGER a cărei sintaxă este:

```
DROP TRIGGER <nume> ON <tabela>[CASCADE|RESTRICT]
```

unde:

nume reprezintă numele unui trigger deja definit.

tabela numele tabelei pentru care se aplică trigger-ul.

Triggerul identificat prin *nume* care este ataşat tabelei specificate este eliminat. Dacă numele său nu se regăseşte în baza de date se generează o eroare. Dacă se specifică opțiunea RESTRICT se respinge executarea comenzii dacă triggerul are ataşate obiecte dependente. Dacă se specifică opțiunea CASCADE toate obiectele dependente se elimină automat.

5.6. BIBLIOGRAFIE

[85] *** – *Mimer SQL - Reference Manual*, Version 8.2; Mimer Information Technology AB, Uppsala, Sweden 2000.

[86] Browne Allen, Balter Alison: *Bazele Access 95*; Editura Teora, Bucureşti 1998.

[87] CAE specification, *Structured Query Language (SQL)*, Version 2. X/Open document number: C449. ISBN: 1-85912-151-9.

[88] Din Akeel I.: *Structured Query Language (SQL) – A practical introduction*; Blackwell,

[89] ISO/IEC 9075:1992(E) *Information technology–Database languages–SQL*: International Standard for the Database Language SQL; ISO 9075:1992.

[90] ISO/IEC 9075-1:1999(E) *Information technology - Database languages – SQL; Part 1: Framework (SQL / Framework)*; ISO 9075-1:1999.

[91] ISO/IEC 9075-2:1999(E) *Information technology - Database languages – SQL; Part 2: Foundation (SQL / Foundation)*; ISO 9075-2:1999.

[92] ISO/IEC 9075-3:1999(E) *Information technology - Database languages – SQL; Part 3: Call-Level Interface (SQL / CLI)*; ISO 9075-3:1999.
[93] ISO/IEC 9075-4:1996(E) *Database Language SQL - Part 4: Persistent Stored Modules (SQL/PSM)*; ISO 9075-4:1996.
[94] ISO/IEC 9075-4:1999(E) *Information technology - Database languages – SQL; Part 4: Persistent Stored Modules (SQL / PSM)*; ISO 9075-4:1999.
[95] ISO/IEC 9075-5:1999(E) *Information technology - Database languages – SQL; Part 5: Host Language Bindings (SQL / Bindings)*; ISO 9075-5:1999.
[96] Luers Tom: *Bazele Oracle*7; Editura Teora, Bucureşti 1998.
[97] Perkins Jeff, Morgan Bryan: *SQL fără profesor, în 14 zile*; Editura Teora, Bucureşti 1998.
[98] Popa Gheorghe, et al.: *Sisteme de gestiune a bazelor de date*; Editura ALL, Bucureşti 1994.

Capitolul 6

BAZE DE DATE DIN MANAGEMENTUL INFRASTRUCTURII

Fiecare instituție implicată în administrarea patrimoniului infrastructurii transportului se confruntă permanent cu starea sistemului în funcțiune, cu performanțele sale, cu modificările în curs şi cu facilitățile viitoare ce urmează a se construi. Facilitățile viitoare depind de nevoile viitoare de transport ale utilizatorilor şi de capacitatea şi dorința acestor utilizatori de a plăti prețul pentru aceste facilități.

Gestionarea eficientă a infrastructurii transporturilor presupune existența unei infrastructuri informaționale corespunzătoare la nivelul administratorului. Acesta trebuie să posede aparatura necesară pentru a putea colecta, transmite, stoca, regăsi, prelucra şi raporta datele legate de inventar dar şi instrumentele logice şi metodologice de procesare a acestora şi obținerea de informații utile.

O administrare promptă şi corectă se poate baza numai pe date corecte şi disponibile atunci când sunt cerute. Aceste date trebuie colectate şi păstrate în baze de date. Prelucrarea lor trebuie să genereze strategii utile şi utilizabile care să conducă la păstrarea sau îmbunătățirea stării globale a sistemului de transport. Permanenta evoluție a tehnologiei transporturilor, modificările din mediul economic şi social impun modificări în necesarul de date utile.

Modalitățile de colectare, transmitere, stocare, regăsire şi diseminare a datelor sunt permanent afectare de evoluția tehnologiei informației. În consecință, procedurile de evaluare a necesarului de date, metodele de colectare, structura bazelor de date, programele de prelucrare şi metodologia de diseminare trebuie să fie flexibile, adaptabile cu costuri minime la noile condiții.

6.1. NECESARUL DE DATE

TIPURI DE DATE NECESARE

Deoarece datele de detaliu se pot schimba, tratarea ansamblului trebuie să se facă la nivel de categorie. Pentru cunoaşterea completă a sistemului trebui să dispunem de patru categorii de date:

- Ofertă;
- Cerere;
- Performanțe;
- Impact.

Oferta descrie datele care se referă la rețeaua fizică şi la serviciile furnizate în manieră comercială precum şi costurile acestora. *Cererea* include date asupra nevoii de mobilitate a bunurilor şi persoanelor, distanțele parcurse, frecvența utilizări şi costurile asociate. Datele privind *performanța* se constituie într-o măsură a capacității ofertei de a acoperi cererea cu costuri care să asigure eficiența economică. Datele de *impact* descriu efectele pe care sistemul de transport le produce asupra mediului înconjurător fizic şi social.

Stocarea datelor trebuie să țină cont de categoria de transport la care se referă. Datele adresează specific căile rutiere, calea ferată, transportul aerian sau naval. Unele organisme legislative şi guvernamentale din diferite țări tratează sistemul de transport în comun ca un element separat şi se impune evidențierea individualizată a acestuia.

În cadrul categoriilor de date enumerate mai sus sunt incluse următoarele tipuri de date:

❖ **Oferta**

- Rețeaua rutieră:
  - Date de sistem: intersecții şi sectoare; lungime şi număr de benzi; capacitatea; împărțirea pe clase funcționale şi categorii de administrare; suprafața de teren ocupată; variante ocolitoare; etc.
  - Date privind serviciile: acces din diferite locații şi conexiunea dintre diferite categorii de drumuri; acces intermodal la punctele de transfer (CF, aeroport, port cu distanțe corespunzătoare); furnizori de servicii de depanare şi informare; taxe (de trecere, de parcare etc.);
  - Date privind utilitățile: inventarul utilităților (stații de autobuz, zone de parcare, terminale de camioane; zone de transfer a containerelor, etc.); date de utilizarea terenului de către utilități; zone urbane permise pentru accesul camioanelor; etc.
  - Date de stare: starea îmbrăcăminților rutiere, pe categorii de drum (IRI, degradare, etc.); date privind starea structurilor rutiere (poduri,

tuneluri, rampe de acces, ziduri de sprijin, drenuri etc.); vârsta sectoarelor şi structurilor rutiere; etc.
  - Proiecte: proiecte propuse şi aprobate a fi executate cu specificarea surselor de finanţare; planuri de dezvoltare şi modificare a reţelei; date de istoric privind lucrările efectuate şi evoluţia proiectelor (întreţinere, reparaţii, reabilitare, inspecţie etc.); proiecte de întreţinere; proiecte tip (de construcţie, întreţinere, reparare, reabilitare, înlocuire).
- Calea ferată:
  - Date de sistem: lungimea liniei (cu destinaţie de transport pasageri şi mărfuri); noduri şi sectoare; capacitate; împărţirea pe categorii şi tipul de tractare permis; teren utilizat; etc.
  - Date privind serviciile: localităţi deservite; frecvenţa şi orarul curselor de pasageri; disponibilităţi pentru transport mărfuri; furnizorii de servicii; taxe şi costuri de călătorie; etc.
  - Date privind utilităţile: număr de vagoane disponibile; inventarul echipamentelor pentru trecerile la nivel cu drumurile; terminale pentru transfer intermodal de mărfuri şi călători; depozite disponibile; inventarul infrastructurii; etc.
  - Date de stare: vârsta liniei, vagoanelor, podurilor, tunelurilor şi a echipamentului; înregistrări privind întreţinerea; etc.
  - Proiecte: lista proiectelor realizate, lista proiectelor noi propuse pentru realizare, extinderi şi modificări, proiecte de întreţinere etc.
- Transport aerian:
  - Date de sistem: aeroporturi şi piste oferite, terenul utilizat pentru extinderea aeroporturilor, capacitatea şi utilizarea principalelor linii;
  - Date privind serviciile: număr de furnizori şi localizarea lor, locaţii deservite, frecvenţa transporturilor, acces la facilităţi de transfer şi transfer intermodal, tarife şi componenţa lor etc.;
  - Date privind utilităţile: facilităţi de transfer a pasagerilor, echipamente de transfer marfă, spaţii de depozitare a mărfurilor etc.;
  - Date de stare: starea pistelor de aterizare (vârstă, întreţinere, istoricul intervenţiilor), starea terminalelor (vârstă, întreţinere, istoricul intervenţiilor), starea avioanelor (vârstă, întreţinere, istoricul reparaţiilor), starea echipamentelor de transfer etc.;
  - Proiecte: lista proiectelor realizate, lista proiectelor noi propuse, date de evaluare a proiectelor, extinderi şi modificări, proiecte de întreţinere etc.
- Transport naval:
  - Date de sistem: porturi existente, porţiunea de teren ocupată pe uscat, docuri şi capacitatea lor, capacitatea principalelor linii, teren disponibil pentru extinderea porturilor;

- Date privind serviciile: acces spre alte moduri de transport, linii navale şi linii de feribot, linii de barje, conexiuni multimodale, perioada din an cât portul este practicabil, valoarea şi compoziţia tarifelor;
- Date privind utilităţile: număr de furnizori, număr de spaţii de acostare pentru nave particulare, spaţii de depozitare a mărfurilor, facilităţi de transfer, locuri de acostare, etc.
- Date de stare: programul de dragare, docuri şi acostamente (vârstă, întreţinere, istoricul intervenţiilor), mijloace de dirijare a navigaţiei (vârstă, întreţinere, istoricul intervenţiilor), nave şi feriboturi (vârstă, întreţinere, istoricul intervenţiilor), adâncimea şi lăţimea canalelor, etc.
- Proiecte: lista proiectelor realizate, lista proiectelor noi propuse, date de evaluarea proiectelor, extinderi şi modificări planificate, proiecte de întreţinere etc.

❖ **Cererea**

- Date economice: date de venit pe gospodării şi regiuni (istoric, prezent, prevăzut), date privind angajarea forţei de muncă (istoric, prezent, prevăzut), posesia de vehicule, costul pe călătorie, depozite şi distribuitori, puncte de import/export, proiecte de dezvoltare economică;
- Date demografice: date privind populaţia şi forţa de muncă, caracteristicile gospodăriilor, etc.
- Utilizarea terenului: parcelare (destinaţie şi suprafeţe – istoric, prezent, perspectivă), date privind gospodăriile (localizarea, distribuţie, densitatea ocupării), date privind accesul, zonarea etc.;
- Date privind circulaţia: date privind necesarul de transport origine-destinaţie, factori multimodali, factori speciali de generare a transportului (turism, excursii profesionale, pelerinaje, evenimente speciale), date privind volumul de trafic, factori decizionali privind alegerea modului expediţie şi deplasare (timp de parcurs prin diferite moduri, costurile relative);
- Comportamentul utilizatorilor: date privind opţiunile în transport (auto, naval, aerian, cale ferată, costuri de transport, costuri de parcare, costuri de închiriere, procent de zone industriale şi comerciale accesibile prin mers pe jos), date psihologic privind comportamentul în alegerea, date privind preferinţele (disponibilitatea de a plăti, nivel de performanţă cerut, disponibilitatea de a coopera în realizarea deplasărilor), comportamentul transportatorilor (respectarea orarului, preţuri discriminatorii, contracte multimodale).

❖ **Performanţa**

- Siguranţa transporturilor: date privind incidentele (număr, tip, localizare şi durată), date privind accidentele (număr, tip, localizare şi durată), date privind intervenţiile medicale (timp de răspuns, număr de furnizori etc.)
- Măsura performanţei: date privind performanţele şoselele (congestii, deplasări tonă×kilometru, întârzieri, viteza medie de deplasare, localizarea

accidentelor şi timpul mediu de degajare – istoric, prezent, perspectivă), costul transportului, timp de livrare, procent de degradare a mărfurilor, timp de aşteptare la terminale, etc.;

❖ **Impactul**

- Calitatea aerului: date privind vehiculele înregistrate (tip, număr, tipul de combustibil consumat), date privind viteza pe clase de vehicule, date privind numărul de călătorii, emisii de poluanți pe clase de vehicule (istoric, prezent, perspectivă);
- Alte date privind mediul: impact vizual şi estetic, zgomot şi vibrații emise, date privind ecosistemele traversate (animale şi plante rare sau ocrotite, suprafața zonelor afectate de infrastructura transporturilor), situri arheologice afectate sau blocate, rezervații;
- Utilizarea terenului: impact socio-economic, impact regional;
- Energia: cantitatea de energie consumată prin fiecare mod de transport, eficiența energiei consumate, prețul energiei consumate;
- Creştere economică: impactul asupra forței de muncă, impactul asupra dezvoltării economice regionale, acces la resursele naturale, acces la piețele interne şi externe etc.

NECESARUL DE DATE LA DIFERITE NIVELURI DE DECIZIE

Pentru a asigura un management competitiv este necesar a se procesa date atât la nivel de rețea cât şi pentru locații punctuale pentru proiecte individuale. La fiecare se iau anumite tipuri de decizii. Este deci natural ca pentru fiecare nivel de decizie să existe nevoi diferite de date. În practică se întâlnesc trei niveluri de management: nivelul managementului rețelei, nivelul programelor de rețea, nivelul managementului proiectelor. Fiecare utilizează seturi proprii de date şi generează propriile seturi de date şi informații de ieşire.

Nivelul managementului rețelei se referă la decizii strategice. La acest nivel se utilizează pentru:

- Proiecția cerințelor viitoare ale încărcării de trafic pe rețea;
- Evaluarea stării prezente a rețelei;
- Predicția stării viitoare la nivelul global al rețelei în diverse variate strategice: întreținere, reabilitare şi reconstrucție;
- Evaluarea impactului diverselor strategii asupra siguranței traficului, eficienței economice şi mediului înconjurător;
- Calculul bugetelor asociate diverselor strategii pentru viitoarele intervale financiare;
- Identificarea strategiilor optimale dezvoltare a rețelei;

- Realizarea de analize de alocare a costurilor şi studii de tendinţă privind generarea de venituri bugetare.

Nivelul programelor de reţea este zona operaţională managementului. Aici se acumulează şi se prelucrează date pentru:

- Determinarea istoricului şi tendinţelor secţiuni din reţea;
- Estimarea efectelor factorilor climatici şi de mediu asupra materialelor;
- Monitorizarea performanţelor: viabilitate, siguranţă, capacitate structurală;
- Evaluarea stării componentelor reţelei şi stabilirea nevoii de lucrări de intervenţie (întreţinere, reparaţii, reabilitare, reconstrucţie);
- Predicţia performanţelor viitoare ale elementelor componente ale reţelei;
- Prioritizarea lucrărilor de intervenţie pe zone şi categorii de intervenţie;
- Stabilirea standardelor de performanţă şi productivitate pentru echipele şi echipamentele de întreţinere;
- Programarea activităţilor de întreţinere;
- Determinarea costurilor unitare ale activităţilor de întreţinere, reabilitare şi construcţie;
- Estimarea duratei de serviciu pentru lucrările de întreţinere şi reabilitare.

Nivelul managementului de proiect utilizează date specifice locaţiei şi proiectului în cauză pentru:

- Determinarea caracteristicilor geometrice şi de material ale diferitelor componente;
- Determinarea elementelor de proiectare pentru diferite componente;
- Determinarea duratei de viaţă rămase;
- Selecţia întreţinerii şi metodelor de consolidare;
- Analiza alternativelor de proiect şi selecţia celei mai convenabile.

6.2. LOCUL BAZELOR DE DATE ÎN AGENŢIA DE ADMINISTRARE

Simplul fapt al existenţei unei baze de date nu ne rezolvă automat problemele. Este necesară dezvoltarea unui întreg sistem pentru a evidenţia valoarea bazei de date. Este necesar să dezvoltăm sistemul informatic integrat ca parte a sistemului informaţional. Aceasta înseamnă stabilirea de canale de comunicaţie, paşi de urmat şi calendare pentru furnizarea de date pentru baza de date.

De notat faptul că baza de date este foarte importantă dar este doar o parte a băncii de date. Banca de date fiind acea parte a sistemului informatic în care se primesc, se stochează, se extrag şi se procesează datele generând informaţii semnificative (a se revedea capitolul 3). În acest context este util să avem nu doar o bază de date foarte bună ci un sistem complet capabil să primească, să păstreze, să furnizeze, să prelucreze şi să utilizeze într-o manieră corectă informaţiile rezultate.

O atenţie deosebită trebuie acordată persoanei sau organizaţiei care administrează baza de date în totalitate sau în parte. Este recomandat ca acel compartiment care este cel mai afectat de date să administreze şi să supervizeze baza de date şi să-i asigure eficienţa.

Un exemplu de colectare a datelor este prezentat în figura următoare.

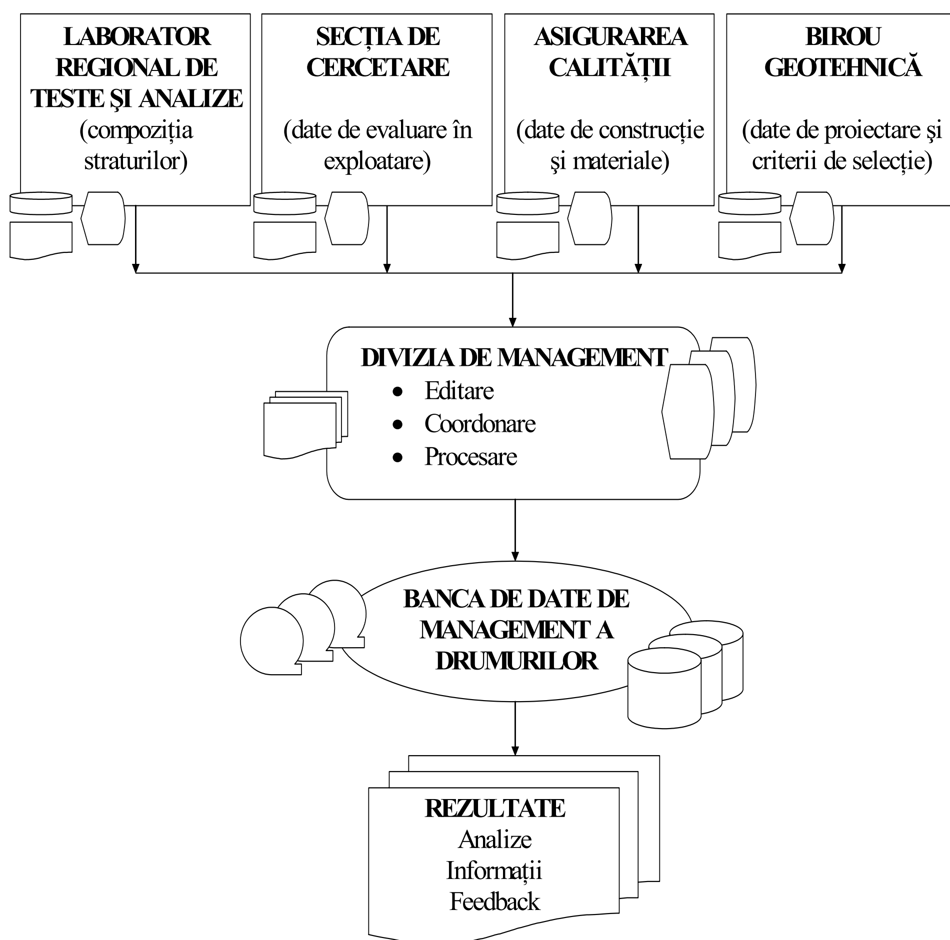


Fig. 34 Colectarea datelor şi banca de date pentru o agenţie de drumuri [102]

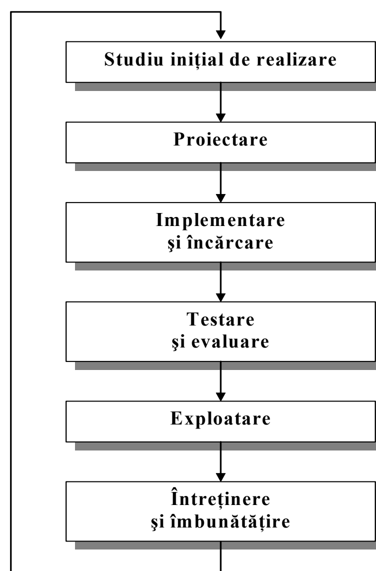


Fig. 35 Ciclul de viaţă al bazelor de date

Un sistem de baze de date, ca orice produs ingineresc, trebuie gândit ca având un ciclu de viaţă [105], descris în Fig. 35.

Acţiunile ce trebuie întreprinse în fiecare din fazele unei baze de date pentru infrastructura transporturilor sunt următoarele:

Studiul iniţial de realizare a bazei de date: analiza inventarului agenţiei; definirea problemelor şi restricţiilor; definirea obiectivelor; delimitarea întinderii.

Proiectarea bazei de date: crearea proiectului conceptual; selectarea sistemului soft de gestionare; crearea proiectului logic; crearea proiectului fizic.

Implementarea şi încărcarea: instalarea SGBD şi crearea bazei de date; încărcarea şi conversia datelor.

Testarea şi evaluarea: testarea bazei de date; calibrarea bazei de date; evaluarea bazei de date şi a aplicaţiilor.

Exploatarea bazei de date: producerea fluxului de informaţii cerut.

Întreţinerea şi îmbunătăţirea: identificarea defectelor; observarea erorilor ascunse; modificare şi îmbunătăţire.

PAŞII DE BAZĂ ÎN DEZVOLTAREA UNEI BAZE DE DATE PENTRU ÎNTREŢINEREA INFRASTRUCTURII TRANSPORTURILOR

Paşii de bază ce trebuie parcurşi în dezvoltarea unui sistem de baze de date pentru întreţinere a infrastructurii rutiere sunt descrişi în Fig. 36. Acest sistem este unul de bază verificat în timp. Urmărirea paşilor trebuie să se facă însă cu prudenţă şi după o analiză atentă a situaţiei specifice a cazului concret.

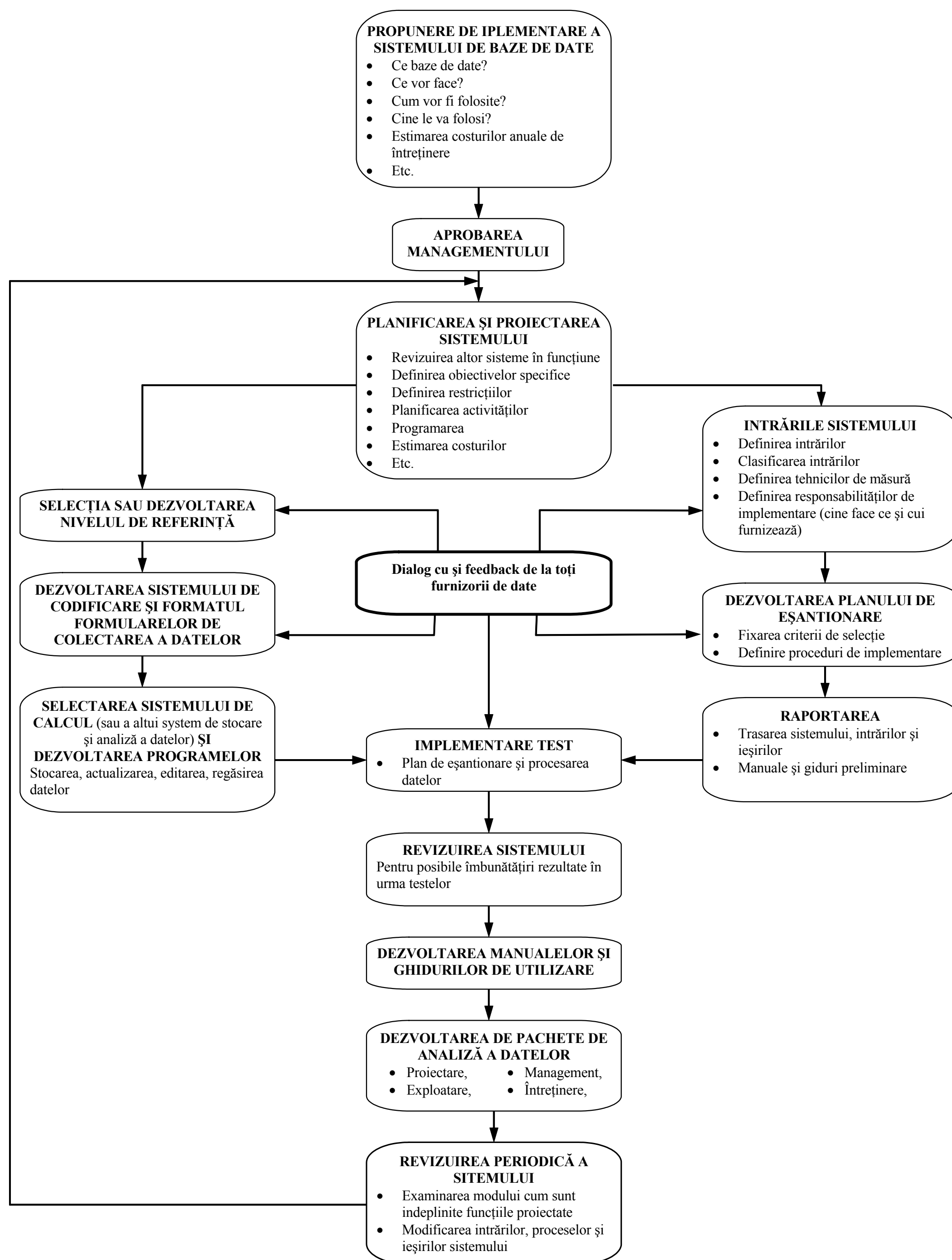


Fig. 36 Paşii în dezvoltarea unui sistem de baze de date pentru drumuri [102]

6.3. IMPLEMENTAREA BAZELOR DE DATE ÎN TRANSPORTURI

AND - BANCA CENTRALĂ DE DATE TEHNICE RUTIERE

Un exemplu în ţara noastră este baza de date privind infrastructura transporturilor a fost dezvoltată în cadrul Administraţiei Naţionale a Drumurilor. Denumită Banca Centrală de Date Tehnice Rutiere – BCDTR, conţine date utilizate în administrarea reţelei de drumuri naţionale din România.

Administraţia Naţională a Drumurilor a trecut încă dinainte de 1990 la crearea unei baze de date care să cuprindă date despre întregul inventar rutier. Banca Centrală de Date Tehnice Rutiere – BCDTR a fost concepută ca o structură dinamică.

Concepută printr-un efort naţional, BCDTR a fost iniţial implementată la INCERTRANS, la vremea respectivă singurul institut de profil din ţară. Ea a fost preluată şi continuată odată cu înfiinţarea Centrului de Studii Tehnice şi Informatică – CESTRIN (ca organism tehnic al AND) care administrează şi date stocate.

Elaborată iniţial în anii optzeci pe mini-calculatoare din seria CORAL, banca de date a fost translatată pe micro-calculatoare compatibile IBM-PC care rulează sub Windows. Sistemul de gestiune al bazelor de date ales este ORACLE. Concepută de la început ca un sistem mare, ea a fost creată utilizând sistemul de gestiune ORACLE 2.0, o implementare pentru mini-calculatoare şi a preluat abilităţile şi limitările programului. Pe măsură ce trecerea timpului aducea noi versiuni ale softului de bază, ea a fost trecută pe aceste versiuni şi actualmente este rulată pe reţele de microcalculatoare sub ORACLE8. BCDTR permite adăugarea de tabele noi şi completarea celor existente cu noi coloane.

Datele şi informaţiile au fost astfel structurate încât să descrie cât mai bine reţeaua. Analizând capacitatea de colectare a administraţiei şi colaboratorilor s-a stabilit un nucleu de date de bază urmând ca pe măsură ce se înaintează în gradul de cunoaştere numărul şi tipul de date să se extindă.

Prezentăm mai jos schematic o parte dintre tabelele conţinute de BCDTR aşa cum au fost concepute de specialiştii CESTRIN care au lucrat la dezvoltarea sa.

Date generale şi organizatorice

CODUNIT	Codificarea unităţilor şi subunităţilor administrativ teritoriale
DRDP	Unităţi administrative din cadrul A.N.D.
LOCALIT	Localităţi traversate – date generale
ANEXEAND	Construcţii anexe aparţinând A.N.D.
DEPMAT	Depozite materiale
DEPSCULE	Depozite scule
DUTILAJE	Depozit utilaje
DJDP	Direcţii judeţene de drumuri şi poduri
POSTCINT	Posturi fixe de cântărire

Date generale privind rețeaua de drumuri

DRUMURI	Drumuri publice
DRUMURIE	Drumuri europene
DRUMNAT	Lungime drumuri naționale
DNJUD	Rețea drumuri naționale pe județe
LREALA	Lungimea reală a drumurilor
SUPRAP	Suprapunere sectoare de drum

Date privind sectoarele rutiere

RETDRDP	Rețeaua de drumuri naționale aparținând direcțiilor
RETSECT	Rețeaua de drumuri naționale aparținând fiecărei SDN
RETDISTR	Rețeaua de drumuri naționale aparținând fiecărui district
RETJUD	Rețeaua județeană de drumuri

Date privind

NRBENZI	Număr benzi de circulație pe drumuri
NRBENZIA	Număr benzi de circulație pe autostrăzi
BENZIINC	Benzi de încadrare
BENZIS	Benzi suplimentare la drumuri şi autostrăzi
INTERSECȚII	Intersecții cu alte drumuri
INTERSCF	Intersecții cu calea ferată

Date privind geometria

CURBE	Curbe în plan de situație
DECLIV	Declivități drumuri
PROFTRS	Tip profil transversal
RACVERT	Curbe de racordare verticală

Date privind structura drumurilor

PLATFORM	Platforma drumului
SISTEMR	Alcătuirea sistemului rutier inițial şi a straturilor de ranforsare

Date de siguranța circulației

INDRUT	Indicatoare rutiere
MARCAJE	Marcaje rutiere
SPERIC	Sectoare periculoase cu concentrare de accidente

Date privind elemente adiacente drumurilor

ACOSTAMENTE	Date privind caracteristicile acostamentelor
PCICL	Piste pentru ciclişti
SANTURI	Şanțuri laterale
TALUZURI	Taluzuri împădurite sau protejate
PARAPETE	Stâlpi de dirijare şi parapete
PLANTRUT	Plantații rutiere existente

Date despre poduri

COD POD	Codificare elemente poduri
PODURI	Date generale poduri
PODINAM	Rezultatele încercărilor dinamice la poduri
PODINFR	Caracteristici ale infrastructurii
PODREP	Lucrări de reparaţii efectuate la poduri
PODSTARE	Stare tehnică poduri
PODSTAT	Rezultatele încercării statice efectuate la poduri
PODSUPR	Elemente geometrice şi constructive suprastructură
PODETE	Podeţe

Date despre alte lucrări de artă

TUNELURI	Tuneluri rutiere
ZSPRIJIN	Ziduri de sprijin
DRENURI	Drenuri

Date de relief şi mediu

PAMINT	Natura pământului de fundaţie
RELIEF	Condiţii de relief
TIPCLIM	Tipul climatic
TORENTI	Amenajare torenţi
STABILIT	Stabilitate teren
S_INUD	Sectoare inundabile

Date de stare

ST_TRONSON	Stare tehnică tronsoane omogene
TRONSOANE	Tronsoane omogene
VIABDR	Starea de viabilitate a drumurilor

Date de lucrări de întreţinere

ACTIARNA	Viabilitatea drumului pe timpul iernii
TRATAMENTE	Tratamente bituminoase
RANFORS	Ranforsări necesare

Date de autorizare

LUCRARI	Autorizaţii de lucrări

După cum se observă, în prezent, tabelele implementate nu acoperă în totalitate tipurile de date identificate mai înainte şi nici necesarul de date la fiecare nivel managerial.

Totuşi, aceste date acoperă în bună măsură necesarul de date utilizate în mod curent de Administraţia Naţionale a Drumurilor şi constituie un punct de plecare în eventuale dezvoltări ulterioare, mai performante.

6.4. BIBLIOGRAFIE

[99] *** – *NCHRP Report 401 – Guidance for Managing Transportation Planning Data*; National Research Council, Transport Research Board, National Academy Press, Washington DC, USA, 1997.

[100] *** – *Pavement Design and Management Guide*; Transportation Association of Canada, Ottawa1997.

[101] Haas Ralph, Hudson Ronald, Zaniewski John: *Modern Pavement Management*; Krieger, Malabar-Florida, 1994.

[102] Haas Ralph, Hudson W. Ronald, *Pavement Management Systems*; McGraw-Hill Book Company; 1978.

[103] Huang, Yang H.: *Pavement Analysis and Design*; Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1993.

[104] Hudson, W. Ronald, Waheed Uddin, Ralph C. Haas: *Infrastructure Management: Integrating Design, Construction, Maintenance, Rehabilitation, and Renovation*; McGraw Hill, 1997.

[105] Rob Peter, Coronel Carlos, *Database Systems – Design, Implementation, and Management*, Course Technology Inc., Cambidge MA, 1997.

[106] Scînteie Rodian: *Aspecte privind prelucrarea datelor în analiza infrastructurii rutiere*; Chişinău, 2002.

[107] Shahin, M.Y.: *Pavement Management for Airports, Roads and Parking Lots*; Chapman & Hall, New York 1994.

Capitolul 7

ALGORITMI ÎN MANAGEMENTUL INFRASTRUCTURII

7.1. NOŢIUNI FUNDAMENTALE

Noţiunea de algoritm este fundamentală în domeniul programării, cercetării operaţionale, a analizei de sistem precum şi a modelării şi simulării, a ingineriei în general. Algoritmul este o metodologie sau o procedură ce cuprinde o secvenţă de instrucţiuni înlănţuite logic ce trebuie să rezolve o problemă dată. Algoritmul primeşte un set de date de intrare, le prelucrează şi le transformă un set de date de ieşire bine definite (Fig. 37). Gândirea umană a lucrat în permanenţă pe baze algoritmice chiar dacă nu le-a teoretizat din totdeauna.

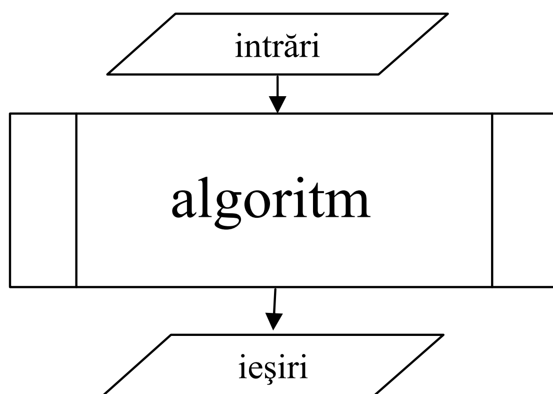


Fig. 37 Algoritm

Matematic, un algoritm este o reprezentare a funcţiei $e = f(i, p)$, unde $i \in I$ este setul de date de intrare, iar $e \in E$ este setul de date de ieşire. Variabila întreagă p se referă la faptul că algoritmul se încheie în p paşi. Valoarea reală a lui p este în general legată de dimensiunea setului de date de intrare. Un algoritm trebuie construit în jurul a patru concepte fundamentale: să fie corect, să fie finit, să fie definit şi să fie eficient.

Corectitudinea: algoritmul trebuie să rezolve corect funcția pentru care a fost proiectat.

Finititudine: secvența de instrucțiuni trebuie să genereze un rezultat după un număr finit de paşi pentru seturile de date care respectă restricțiile impuse.

Definire: paşii trebuie precis definiți şi să efectueze operații precise.

Eficiență: secvența de instrucțiuni (paşii) algoritmului trebuie să poată fi executați pe o maşină „fizic realizabilă”.

Algoritmul este o „rețetă” care să ne conducă de la punctul de pornire la rezultat. Altfel spus un algoritm este o procedură bine definită de calcul matematic, logic şi simbolic care transformă datele de intrare în date de ieşire sau informații. În utilizarea lor algoritmii trebuie să răspundă la câteva întrebări practice:

- Finalitatea: după parcurgerea unui număr de paşi se opreşte şi generează un rezultat. Care sunt restricțiile care se impun şi în ce condiții nu are finalitate?
- Precizia: corectitudinea rezultatului este evaluată cu o anumită toleranță. Acolo unde nu există metode de calcul exacte trebuie stabilite aprioric abaterile tolerate. Ce precizie trebuie adoptată şi care sunt cele mai eficiente metode pe care proiectantul algoritmului trebuie să le aibă în vedere pentru limitarea efectelor propagării erorilor de calcul?
- Viteza: datele de intrare sunt reprezentate de seturi variabile. Durata de parcurgere a tuturor paşilor depinde de dimensiunea seturilor de date de intrare. Structura şi dimensiunea seturilor de date, succesiunea paşilor de calcul şi precizia sunt împreună valori critice. Parcurgerea algoritmului trebuie să se realizeze într-un interval de timp suficient de scurt pentru ca rezultatul generat să mai fie util.
- Spațiul: datele de intrare, structurile ce păstrează valorile intermediare, variabilele de stare, datele de ieşire se regăsesc fizic prin memoria ocupată în calculator. Reprezentarea numerelor reale, de exemplu, poate fi făcută în diferite moduri, fiecare modalitate ocupând un spațiu diferit care uneori depinde şi de construcția fizică a calculatorului. Memoria disponibilă este limitată. În consecință implementarea algoritmului poate depinde de maşina fizică disponibilă.

Paşii care apar pe durata execuției unui algoritm pot fi grupați în:

- Paşi de atribuire (valorile iniţiale sunt ataşate unor variabile);
- Paşi aritmetici (care conţin operaţii simple de adunare, scădere, înmulţire, împărţire sau funcţii complexe ori operaţii simbolice);
- Paşi logici (cum ar fi comparare a două numere etc.).

Din punct de vedere istoric, algoritmii sunt, ca denumire şi ca utilizare, o invenţie recentă chiar dacă actualmente pare o noţiune firească, fără de care nu putem concepe activitatea curentă. Un element cheie în dezvoltarea gândirii algoritmice a fost sistemul poziţional de notare a numerelor. Sistemul de notare numerică roman era deosebit de greoi şi nu permitea dezvoltarea unei gândiri logice, algoritmice. Primul sistem de notare numerică poziţională despre care se păstrează mărturii se pare că a fost utilizat de mayaşi acum 2000 de ani, pe baza unui sistem de numeraţie în baza 20, dar nu există dovezi că s-au dezvoltat şi algoritmi matematici. Chiar dacă au existat lucrări pe această temă ele au fost distruse, împreună cu întreaga literatură ştiinţifică, odată cu cucerirea spaniolă creştină. Fenomene asemănătoare se cunosc şi în Extremul Orient, de exemplu China. De altfel este foarte posibil ca şi în Europa şi Orientul Mijlociu să se fi produs astfel de distrugeri sistematice atunci când noi curente politice şi/sau religioase s-au impus. Eliminarea cărţilor este, în istorie, la fel de răspândită ca şi eliminarea oamenilor.

Sistemul poziţional de notare a apărut în India în jurul anului 600 d.C. şi începe să se răspândească pe la 750 d.C. când după cucerirea musulmană mai multe lucrări indiene ce cuprind descrierea unor algoritmi aritmetici sunt preluate în Persia şi traduse în arabă.

Prima persoană cunoscută că a tratat acest subiect este al-Khwarizmi şi se pare că de aici provine şi denumirea de „algoritm”. Pe la 1200 lucrarea a fost tradusă în latină şi s-a făcut cunoscută în Europa.

Sistemul zecimal a fost inventat în secolul 10 de matematicianul sirian al-Uqlidisi din Damasc. Munca sa a fost reluată şi dezvoltată cinci secole mai târziu de matematicianul persan al-Kashi.

De atunci, proiectare, implementare şi utilizarea algoritmilor a devenit o parte implicită a cercetării ştiinţifice. Chestiunea a fost reluată la nivelul teoretic mai ales după apariţia calculatoarelor când realizarea de programe informatice se bazează larg pe dezvoltarea de algoritmi, iar rezolvarea de probleme complexe implică un consum ridicat de memorie şi de timp, reducerea consumului de resurse fiind un element cheie în acceptarea uneia sau alteia dintre soluţii.

Utilizarea algoritmilor este uzuală în cele mai diverse ramuri ştiinţifice şi tehnice. Studiile pentru rezolvarea de probleme de interes deosebit au generat o serie de algoritmi celebri:

- Construcţiile lui Euclid,
- Calculul lui Newton pentru găsirea rădăcinii,
- Transformata Fourier rapidă,

- Algoritmi de compresie (Huffman, Lempel-Ziv, GIF, MPEG),
- Algoritmi de criptare,
- Algoritmul simplex pentru programare liniară,
- Algoritmi de găsire a căii minime (Dijkstra, Bellman-Ford etc.),
- Controlul congestiei şi redirectarea traficului,
- Recunoaşterea formelor (genetică, evaluarea stării de degradare a betonului),
- Algoritmul de triangulaţie (proiectare asistată de calculator, simulare).

7.2. DESCRIEREA ALGORITMILOR

Pentru a se proiecta eficient algoritmii în aşa fel încât să nu existe probleme de înţelegere pentru cei care fac efectiv programarea s-au inventat metode de reprezentare. Cele mai cunoscute sunt descrierea prin diagrame logice şi limbajul pseudocod. Acestea sunt instrumente utile şi foarte uşor de manevrat.

DIAGRAMELE LOGICE

Diagramele logice constau în reprezentarea grafică prin blocuri a operaţiilor care trebuie efectuate. Blocurile împreună cu conectorii şi cu săgeţile de legătură, care stabilesc fluxul datelor, determină modalităţile de prelucrare a datelor de la intrare până la ieşire.

Tabelul 6. Elemente utilizate în schemele logice

	Bloc de intrare/ieşire
	Bloc de proces
	Bloc decizional
	Bloc al proceselor predefinite
	Bloc de pregătirea
	Conectori

LIMBAJUL PSEUDOCOD

Pseudocod este un crochiu al unui algoritm, scris într-o formă care poate cu uşurinţă să fie transformat în program.

Pseudocod-ul nu poate fi compilat sau executat şi nu are reguli reale de sintaxă sau de formulare. Este doar o înşiruire de paşi logici de urmat în rezolvarea problemei. Permite analistului şi programatorului să se concentreze asupra problemei şi fără să insiste asupra considerentelor legate de limbajul de programare.

Descrierea algoritmului cu ajutorul pseudocod se poate face fără a cunoaşte de la început în ce limbaj de pragramare va fi scris codul, ba chiar se poate face fără a cunoaşte obligatoriu un limbaj de programare. De asemenea, nu există restricţii asupra limbii utilizate, dar cei mai mulţi analişti preferă direct limba engleză. Sunt acceptate semne grafice sugestive împrumutate sau nu din formulele matematice. Astfel pentru atribuire se utilizează frecvent semnul ← (săgeată spre stânga).

7.3. EFICIENŢA ALGORITMILOR

Eficienţa unui program rezultat din implementarea unui algoritm depinde de tehnicile utilizate. Putem distinge aici două noţiuni: (1) algoritm „conceptual" şi (2) algoritm „complet".

Algoritmul conceptual se limitează la a privi lucrurile la modul general fără a intra în detalii. Cel de al doilea, pornind de la primul, intră în amănunte şi tratează inclusiv reprezentarea şi accesul la date, impunând alegerea unor anumite structuri specifice şi a tehnicilor particulare de tratare a problemei.

COMPLEXITATEA ALGORITMILOR

Aşa cum s-a precizat mai sus un algoritm este o procedură de rezolvare secvenţială a problemelor care apar în inginerie şi nu numai. Timpul de calcul consumat diferă de la un caz particular la altul. Pentru anumite situaţii favorabile timpul de calcul poate fi extrem de scurt. Pentru alte situaţii nefavorabile trebuie să aşteptăm foarte mult pentru ca algoritmul să se încheie. Deoarece, pentru rezolvarea unei anumite probleme, pot fi concepuţi mai mulţi algoritmi de calcul este bine să găsim o măsură a performanţei fiecăruia.

Utilizând o astfel de măsură şi comparând costurilor şi beneficiilor generate putem alege cel mai bun dintre algoritmii disponibili pentru rezolvarea problemei.

Cea mai des folosită măsură este timpul de calcul. Uneori se poate utiliza drept indicator şi cantitatea de memorie utilizată. Aceasta din urmă este tot mai rar folosită, mai ales ca termen ajutător, deoarece calculatoarele disponibile oferă memorie suficientă pentru rezolvarea problemelor practice. Totuşi, acesta este un factor de care trebuie ţinut seama în probleme de conducere "on-line" a proceselor,

acolo unde costul permis pentru echipamentul achiziționat este restricționat de beneficiile predictibile.

De remarcat că simpla comparare a timpului de calcul prin implementarea şi rularea algoritmilor, deşi furnizează unele indicații, nu este neapărat eficientă. Timpul efectiv de rulare este diferit pe diferite calculatoare. De asemenea este dependent de setul particular de date, de limbajul de programare şi de compilatorul specific folosit.

Este necesară o altfel de metodă care să ne ofere o măsură mai precisă şi să nu fie influențată de situații particulare (calculator, compilator, valori etc.). Cea mai sigură metodă este calcularea numărului total de paşi pe care un algoritm îi parcurge pe durata rezolvării problemei şi care determină timpul total de calcul. Acesta depinde de dimensiunea setului de date de intrare. Funcția care stabileşte o relație între dimensiunea setului de date de intrare şi numărul de paşi se numeşte **complexitatea** algoritmului. Din nefericire nu se poate totdeauna stabili şi demonstra o relație exactă între cele două mărimi.

STRUCTURI CICLICE

În cazul paşilor aritmetici şi de atribuire problemele sunt clare: fiecare pas contează o singură dată. Probleme apar în cazul paşilor logici. Aceştia pot conduce la executarea de secvențe de instrucțiuni de lungimi diferite funcție de valoarea rezultată în operația logică. De asemenea prin ataşarea unei instrucțiuni de salt înapoi pot rezulta structuri ciclice în desfăşurarea algoritmului.

Structuri ciclice tipice sunt *while* , *repeat...until*.

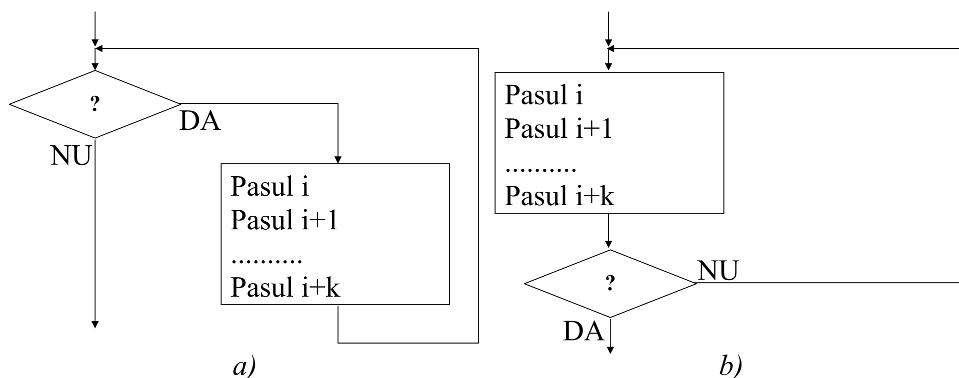


Fig. 38 Structuri ciclice tipice: while (a) şi repeat...until (b)

Se poate observa că paşii cuprinşi într-o buclă de tip *repeat...until* se execută cel puțin o dată şi acest lucru trebuie considerat când se studiază algoritmul.

Un exemplu tipic de algoritm care utilizează o structură ciclică de program este calculare factorialului unui număr (Fig. 39).

```
Funct Factorial( n: Integer ) : Integer

If n=0 Then Return 1
Rez ← 1
i ← 1
Repeat               // început buclă
    Rez ← Rez * i
    i ← i + 1
Until i=n            // sfâşit buclă
Return Rez

EndFunct
```

Fig. 39 Calculul funcției factorial prin utilizarea structurilor ciclice

Un alt exemplu tipic este calculul şirului Fibonacci. Acest şir este caracterizat prin următoarele: primul număr este $F_0 = 0$, următorul este $F_1 = 1$ iar celelalte numere reprezintă suma precedentelor două din şir.

```
Funct Fibonacci( n: Integer ) : Integer

If n<0 then Error
If n=0 Then Return 0
If n=1 Then Return 1

F0 ← 0
F1 ← 1

I ← 2
Repeat               // început buclă
    Temp ← F0 + F1
    Fo ← F1
    F1 ← Temp
    i ← i + 1
Until i = n          // sfâşit buclă

Return F1

EndFunct
```

Fig. 40 Rezolvarea şirului Fibonacci prin structură ciclică

RECURSIVITATEA

Recursivitatea este o metodă de rezolvare problemelor prin descrierea unei funcții prin apelarea aceleiaşi funcții cu valori inferioare.

Exemple pot fi cele date la structuri ciclice. În continuare prezentăm rezolvarea algoritmului pentru calculul factorialului unui număr prin apel recursiv.

```
Funct Factorial( n: ) :

If n=0 Then Return 1
Return n * Factorial(N-1)

EndFunct
```

Fig. 41 Rezolvarea funcției factorial prin apel recursiv

Similar şirul lui Fibonnaci poate fi rezolvat prin apel recursiv. Algoritmul este prezentat în figura următoare.

```
Funct Fibonacci( n: ) :

If n<0 then Error
If n=0 Then Return 0
If n=1 Then Return 1

Return Fibonacci(n-1) + Fibonacci(n-2)

EndFunct
```

Fig. 42 Rezolvarea şirului Fibonacci prin apel recursiv

O remarcă imediată este faptul că recursivitatea simplifică mult munca programatorilor. Programele scrise sunt mult mai scurte. Complexitatea algoritmului, în sensul resurselor consumate, nu respectă totdeauna aceeaşi regulă.

Un exemplu tipic pentru algoritmii rezolvați prin recursivitate este problema turnurilor din Hanoi.

Hanoi este capitala Vietnamului, un oraş cu o istorie milenară. Într-una din mănăstirile sale un grup de călugări sunt păstrătorii unui misterios exercițiu care a preocupat atât pe curioşi cât şi pe oamenii de ştiință. Ei dețin trei turnuri (tije verticale) şi 64 de discuri din aur de diferite dimensiuni. Discurile sunt aranjate pe primul turn de jos în sus în ordine descrescătoare. Mutând câte un disc odată, ele trebuie să ajungă în aceeaşi ordine pe cel de-al treilea turn. Se poate utiliza al doilea turn ca ajutor dar un disc mai mare nu trebuie aşezat niciodată peste unul mai mic.

Călugării trebuie să mute continuu discurile pentru a rezolva problema. Se spune că atunci când toate cele 64 de discuri vor ajunge la locul lor va veni sfârşitul lumii.

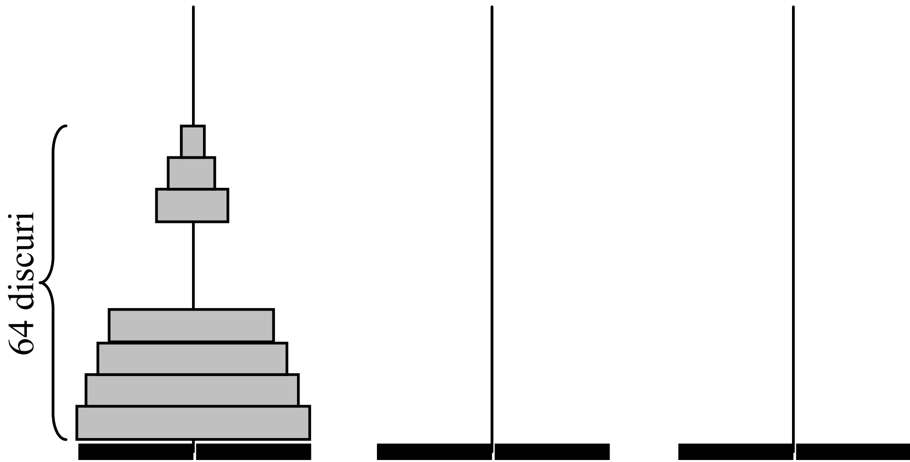


Fig. 43 Turnurile din Hanoi

Problema a fost studiată şi formalizată de matematicianul francez Lucas în 1883.

Pornind de la rezultatele sale s-a putut trasa algoritmul care în formatul recursiv are formă prezentată în următoarea figură:

```
Proc Hanoi ( n: integer; A,B,C )

If (n=1)
Then
   Mută_un_discul_de_pe_A _pe_B
Else
   Hanoi(n-1,A,C,B)
   Hanoi(  1,A,B,C)
   Hanoi(n-1,C,B,A)
Endif

EndProc
```

Fig. 44 Rezolvarea recursivă a problemei turnurilor din Hanoi

Numărul de mutări necesare pentru rezolvarea acestei probleme este dat de următorul raţionament:

$$N_M\left[Hanoi(n,A,B,C)\right]=$$
$$N_M\left[Hanoi(n-1,A,C,B)\right]+N_M\left[Hanoi(1,A,B,C)\right]+N_M\left[Hanoi(n-1,C,B,A)\right]$$

Considerând τ durat unei mutări, timpul de rezolvare este dat de:

$$
\begin{aligned}
T(n) &= 2T(n-1)+\tau \\
&= 2^2 T(n-2)+2\tau+\tau \\
&= 2^3 T(n-2)+2^2\tau+2\tau+\tau \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&= 2^{n-1}T(1)+2^{n-2}\tau+\cdots+2^2\tau+2\tau+\tau \\
&= \left(2^{n-1}+2^{n-2}+\cdots+2^2+2+1\right)\tau \\
&= \left(2^n-1\right)\tau
\end{aligned} \tag{5}
$$

Dacă $n = 64$ atunci numărul de paşi de urmat este

$$N = 2^{64} - 1 = 18.446.744.073.709.551.615 \tag{6}$$

Presupunând că o mutare durează o secundă, procesul va dăinui în total 584.942.417.355 ani. Se pare că lumea nu este în pericol.

7.4. ANALIZA COMPLEXITĂȚII ALGORITMILOR

O trecere în revistă a literaturii de specialitate relevă existența utilizării pe scară largă a trei modalități de măsurare a performanțelor unui algoritm: analiza empirică, analiza cazului mediu şi analiza cazului cel mai defavorabil. Fiecare dintre măsuri are meritele dar şi dezavantajele sale.

METODE DE ANALIZĂ

Analizei empirică are ca obiectiv estimarea comportamentului algoritmului în situații practice. Pe baza algoritmului se scrie un program şi se testează modul de lucru pe clase de situații particulare ale problemei. Abordarea aceasta este utilă în validarea analizei matematice care a condus la realizarea algoritmului. Se pot compara performanțele a doi algoritmi realizați pentru a rezolva aceeaşi problemă prin rularea pe aceeaşi maşină, în aceleaşi condiții de lucru şi cu acelaşi set de date de intrare. De menționat aici că selecția datelor de intrare este extrem de importantă. Pentru seturi particulare diferite se pot obține timpi de rulare diferiți chiar cu acelaşi algoritm în două rulări succesive. Depăşirea impasului se face prin generarea de seturi aleatoare de date care se utilizează pentru ambele programe făcând comparația la nivel global.

Totuşi analiza empirică se poate efectua numai după ce s-a realizat complet şi corect scrierea programului. Invalidarea testului duce la irosirea manoperei pentru programare fără ca aceasta să aibă o finalitate.

Dezavantajele majore ale analizei empirice sunt:

- performanţele algoritmilor depind de cele mai multe ori de limbajul de programare, compilatorul şi computerul utilizate pentru experiment precum şi de capacitatea programatorului;
- analiza însăşi este uneori prea costisitoare şi foarte mare consumatoare de timp;
- compararea poate să nu fie concludentă deoarece anumiţi algoritmi se comportă mai bine pentru o clasă de cazuri particulară şi teste diferite efectuate cu scopuri diferite pot conduce la rezultate opuse.

Analiza cazului mediu se bazează pe o analiză probabilistică a apariţiei unor clase de realizări (instanţe) ale problemei şi constă în calcularea unui estimator al numărului mediu de paşi ai algoritmului. Pe baza acestuia se deduce timpul mediu de rulare al algoritmului.

Ca dezavantaje majore putem enumera:

- analiza depinde crucial de funcţia de distribuţie a probabilităţii alese să descrie apariţia realizărilor problemei, modificări ale funcţiei pot conduce la modificări ale evaluării algoritmului;
- aprecierea distribuţiei probabilităţii este dificil de realizat pentru probleme practice, analistul trebuie să parcurgă un număr mare de situaţii unele imposibil de conceput aprioric;
- analiza este cel mai adesea deosebit de complicată şi necesită cunoştinţe matematice deosebite chiar pentru analiza unor tipuri simple de algoritmi. Acest tip de analiză se bazează pe situaţia medie, „cazul tipic" al intrărilor, dar nu furnizează informaţii despre distribuţia ieşirilor. Unele dintre cazurile atipice pot conduce la durate excepţional de mari de calcul chiar dacă pentru cazul mediu algoritmul se comportă mulţumitor.

Analiza cazului cel mai defavorabil furnizează limita superioară a numărului de paşi pe care îi parcurge algoritmul oricare ar fi realizarea problemei. În această analiză contorizăm numărul maxim posibil de paşi prin care ni se garantează că se ajunge la rezultat în oricare situaţie realizabilă în practică.

Această analiză evită problemele care apar în celelalte tipuri de analiză. Este independentă de sistemul şi mediul de calcul şi este relativ uşor de realizat furnizând o garanţie a plafonului maximal.

Dezavantajul major este că evaluarea performanţelor unor algoritmi depinde de existenţa unor situaţii „patologice" a căror probabilitate de apariţie este neglijabilă, practic nulă.

NOTAŢIA ASIMPTOTICĂ

Pentru măsurarea complexităţii prin timpul de rulare s-au dezvoltat diferite instrumente, fiecare fiind caracterizat prin notaţii specifice. Deoarece contabilizarea

detaliată a numărului de paşi ai algoritmului este complicată, dificilă şi neproductivă, principalul instrument de lucru este notația asimptotică.

Trebuie notate câteva elemente:

- suntem preocupați de setul numerelor naturale { 0, 1, 2, … };
- n este dimensiunea setului de date de intrare;
- *T(n)* este timpul de rulare pentru cazul cel mai defavorabil;

Notația asimptotică se bazează pe metoda inducției matematice. Inducția este utilizată pentru demonstrarea unei anumită formule generale care este cunoscută sau uşor de dedus. Sunt urmate trei etape:

(1) dovedirea condiției inițiale;

(2) presupunerea că formula este valabilă pentru *n* şi

(3) dovedirea valorii de adevăr pentru *n*+1.

Formularea standard este:

Dacă propoziția $P(n)$ (în cazul nostru afirmația că timpul cel mai defavorabil este $T(n)$) este adevărată pentru $n = n_0$ şi dacă pentru orice $n \geq n_0$ $P(n)$ implică $P(n+1)$ atunci propoziția $P(n)$ este adevărată pentru oricare n.

Prezentăm în continuare câteva modalități de descriere a complexității algoritmilor prin notații asimptotice.

Notația Θ (theta mare)

Notând $g(n)$ numărul total de paşi, unde *n* este dimensiunea datelor de intrare trebuie să găsim o funcție $\Theta(g(n)) = f(n)$ pentru care există constantele pozitive c_1, c_2, n_0 astfel încât $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ pentru $\forall n \geq n_0$.

- Dacă $f(n) = \Theta(g(n))$ şi $g(n) = \Theta(h(n))$ a atunci $f(n) = \Theta(h(n))$;
- $f(n) = \Theta(f(n))$;
- $f(n) = \Theta(g(n))$ dacă şi numai dacă $g(n) = \Theta(f(n))$.

Notația O (O mare)

Trebuie să găsim o funcție $O(g(n)) = f(n)$ pentru care există constantele pozitive c, n_0 astfel încât $0 \leq f(n) \leq c \cdot g(n)$ pentru $\forall n \geq n_0$.

- Dacă $f(n) = O(g(n))$ şi $g(n) = O(h(n))$ atunci $f(n) = O(h(n))$;

- $f(n) = O(f(n))$;
- $f(n) = O(g(n))$ dacă şi numai dacă $g(n) = \Omega(f(n))$;
- Dacă $f(n) = \Theta(g(n))$ atunci $f(n) = O(g(n))$.

Notaţia *O* mărgineşte superior creşterea timpului de rezolvare a algoritmului

Notația Ω (Omega mare)

Trebuie să găsim o funcţie $\Omega(g(n)) = f(n)$ pentru care există constantele pozitive c, n_0 astfel încât $0 \le c \cdot g(n) \le f(n)$ pentru $\forall n \ge n_0$.

- Dacă $f(n) = \Omega(g(n))$ şi $g(n) = \Omega(h(n))$ atunci $f(n) = \Omega(h(n))$;
- $f(n) = \Omega(f(n))$;
- $f(n) = \Omega(g(n))$ dacă şi numai dacă $g(n) = O(f(n))$;
- Dacă $f(n) = \Theta(g(n))$ atunci $f(n) = \Omega(g(n))$.

Notaţia *Ω* mărgineşte inferior creşterea timpului de rezolvare a algoritmului

Notația o (o mic)

Trebuie să găsim o funcţie $o(g(n)) = f(n)$ pentru oricare $c > 0$ există o constantă pozitivă n_0 astfel încât $0 \le f(n) < c \cdot g(n)$ pentru $\forall n \ge n_0$.

$\lim_{n\to\infty} f(n)/g(n) = 0$.

- Dacă $f(n) = o(g(n))$ şi $g(n) = o(h(n))$ atunci $f(n) = o(h(n))$;
- $f(n) = o(g(n))$ dacă şi numai dacă $g(n) = \omega(f(n))$;

Notația ω (omega mic)

Trebuie să găsim o funcţie $\omega(g(n)) = f(n)$ pentru oricare $c > 0$ există o constantă pozitivă n_0 astfel încât $0 \le c \cdot g(n) < f(n)$ pentru $\forall n \ge n_0$.

$\lim_{n\to\infty} f(n)/g(n) = \infty$.

- Dacă $f(n) = \omega(g(n))$ şi $g(n) = \omega(h(n))$ atunci $f(n) = \omega(h(n))$;
- $f(n) = \omega(g(n))$ dacă şi numai dacă $g(n) = o(f(n))$;

CONSIDERAȚII PRACTICE

În practică s-a impus tot mai mult utilizarea notaţiei O (O mare – „Big O”). Funcţie de notaţia O algoritmii pot primi o caracterizare lingvistică de tipul celor prezentate în tabelul următor.

Tabelul 7. Tipuri de complexitate

Caracterizare	**Notația O**
Constant	O(1)
Logaritmic	O(log n)
Linear	O(n)
N log n	O(n logn)
Cuadratic	$O(n^2)$
Cubic	$O(n^3)$
Polinomial	$O(n^k)$
Exponențial	$O(2^n)$ sau $O(10^n)$
Factorial	O(n!)
	$O(n^n)$

Pentru avea o imagine mai clară a semnificației complexității unui algoritm să considerăm următorul exemplu ipotetic. Presupunem că avem la dispoziție un calculator care execută fiecare pas al unui algoritm într-o micro-secundă (adică are viteza de 1 milion de instrucțiuni pe secundă). Algoritmul va fi executat şi se va opri după $f(n)$ instrucțiuni atunci timpul de rulare, funcție de dimensiunea datelor de intrare va fi în conformitate cu tabelul următor.

Tabelul 8. Exemplu de timp de calcul

$f(n)$	n=2	n=16	n=256	n=1024	n=1048576
1	1 μs	1 μs	1 μs	1 μs	1 μs
$\log_2 n$	1 μs	4 μs	8 μs	10 μs	20 μs
N	2 μs	16 μs	256 μs	1 ms	1 s
$n \log_2 n$	2 μs	64 μs	2 ms	10 ms	21 s
n^2	4 μs	26 μs	66 ms	1 s	12 zile
n^3	8 μs	4,1 ms	17 s	18 min	366000 ani
2^n	2 μs	66 ms	**10^{63} ani**	**10^{295} ani**	**10^{315639} ani**

Pentru comparație, amintim că din datele cunoscute astăzi soarele va dispărea peste doar 5×10^9 ani. Algoritmii cu complexitate exponențială nu sunt utilizabili în practică. Timpii de calcul îi fac utili doar pentru seturi foarte mici de date. Asupra algoritmilor cu complexitate factorială sau n la n nu insistăm, funcția lor de creştere fiind mult superioară celei exponențiale.

Chiar dacă prezintă dezavantaje evidente, complexitatea exponenţială, factorială sau n la n este inevitabilă la nivelul actual de cunoştinţe. De exemplu, problema comisului voiajor (pe o reţea în care nodurile reprezintă oraşele un vânzător trebuie să viziteze toate oraşele exact o singură dată cu costuri minime, cunoscut fiind costul călătoriei pe fiecare latură a reţelei) are cea mai bună soluţie de complexitate exponenţială.

Notaţia O, şi în general notaţiile de complexitate se aplică problemelor cu seturi de date de intrare suficient de mari. Concluziile ce se pot trage pentru problemele mari nu se aplică obligatoriu şi problemelor cu dimensiune mică a vectorului de intrare. Fiind vor de o notaţie asimptotică rezultatul este valabil numai pentru valori mai mari decât o anumită valoare de prag.

Pentru probleme mici nu ne vom baza niciodată pe notaţia O. Vom testa câteva implementări şi vom măsura timpul de lucru după care vom selecta unul rezonabil pentru clasa de probleme pe care trebui să o rezolvăm în mod practic.

Trebuie avut în vedere efectele particulare ale mărimii reduse a setului de date. De exemplu, un algoritm cu complexitate 1000n este mai scump decât unul n^2 pentru n<1000. alte consideraţii importante la selecţia algoritmilor sunt frecvenţa de utilizare a programului, uşurinţa de scriere a codului, uşurinţa de întreţinere, necesarul de memorie etc.

EXEMPLE DE CALCUL A COMPLEXITĂȚII

Operaţii matriciale

O formă simplă de algoritm, dar esenţială în înţelegerea ideii de complexitate, este calculul produsului a două matrice. Presupunând că avem două matrice pătratice cu dimensiunea $n \times n$ calculul produsului se va face respectând următorii paşi:

	Tipul Operaţiei	Număr de operaţii
`Funct MultMatrix ( A[n,n], B[n,n] )`		
`For i=1 To n Do`	Atribuire	n
`   For j=1 To n Do`	Atribuire	n^2
`      C[i,j]=0`	Atribuire	n^2
`      For k=1 To n Do`	Atribuire	n^3
`         C[i,j]=C[i,j]+A[i,k]*B[k,j]`	Atribuire	$2n^3$
`      EndFor`		
`   EndFor`		
`EndFor`		
`Return C[n,n]`		
`EndFunct`		

Fig. 45 Exemplu de calcul al complexităţii

CLASE DE COMPLEXITATE

Aşa cum am văzut mai sus, o problemă poate fi rezolvată într-un anumit timp a cărui durată depinde de echipamentul de lucru dar şi de dimensiunea setului de date de intrare. Dependenţa de datele de intrare defineşte complexitatea algoritmului care rezolvă problema.

Cunoaşterea complexităţi ne poate da indicaţii privind posibilitatea noastră de a rezolva problema într-un interval de timp rezonabil pe maşinile de care dispunem. Ştiind timpul de rezolvare putem deduce dacă o problemă este tratabilă sau intratabilă la nivelul actual al tehnologiei.

Clasa P de complexitate

Un algoritm este mărginit polinomial când complexitatea în cazul cel mai defavorabil este mărginit de o funcţie polinomială p care depinde dimensiunea setului de intrare n. Funcţia polinomială are forma $p(n)=\sum_{i=0}^{m} a_i n^i$, unde m poate fi oricât de mare dar este finit.

O problemă este mărginită polinomial dacă poate fi rezolvată printr-un algoritm mărginit polinomial.

***P** este o clasă de probleme decizionale care sunt mărginite polinomial*

Se spune despre probleme de clasă P că sunt rezolvabile polinomial.

Trebuie remarcat că nu este obligatoriu ca o problemă de clasă P să poată avea un algoritm de complexitate acceptabilă. Totuşi, dacă o problemă nu este de clasă P este inerent intratabilă.

Clasa NP de complexitate

NP este o clasă de probleme decizionale a căror algoritm aparţine clasei Pdar pentru care o soluţie propusă dată pentru un set particular de date de intrare poate fi verificată într-un timp mărginit polinomial.

*O problemă este non-deterministic polinomială – de clasă **NP** dacă:*

- *Se determină în mod nedeterministic o soluţie;*
- *Corectitudinea soluţiei candidat se poate verifica într-un timp mărginit polinomial.*

Rezolvarea acestei clase de probleme are deci două faze. Prima este euristică, de „ghicire” a unei soluţii. Soluţia nu este obligatoriu unică. De aceea două rezolvări succesive sau paralele pot să fie diferite. A doua fază este deterministică, rezultatul său fiind totdeauna acelaşi, rezolvând problema prin soluţia propusă.

Se spune despre probleme de clasă NP că sunt verificabile polinomial. De asemenea se consideră că problemele de clasă NP sunt probleme ce pot fi rezolvate intr-un timp mărginit polinomial pe o maşină de calcul nedeterministă.

Este foarte dificil uneori de găsit o soluţie pentru problemele de clasă NP. Pentru multe tipuri de astfel de probleme din această clasă se presupune că nu există un algoritm mărginit polinomial care să le rezolve. Totuşi datorită complexităţii lor nu s-a putut dovedi.

Clasa de probleme NP-complete

O problemă decizională este de clasă NP-completă dacă aparţine clasei NP şi fiecare problemă din NP poate fi rapid redusă la aceasta.

Putem spune că problemele NP-complete reprezintă cele mai grele probleme din clasa NP. Dacă reuşim să găsim un algoritm mărginit polinomial pentru a rezolva o problemă NP-completă atunci putem să rezolvăm toate problemele NP într-un timp mărginit polinomial.

7.5. TEHNICI DE PROIECTARE A ALGORITMILOR

În rezolvarea de probleme utilizând algoritmi de calcul se pot utiliza mai multe metode de abordare. Între acestea amintim:

- Reducerea la probleme cunoscute;
- Abordarea egoistă sau lacomă (greedy approach)
- Divide şi cucereşte;
- Programare dinamică;
- Utilizarea de structuri de date mai bune;
- Soluţii probabilistice;
- Soluţii aproximative;

REDUCEREA LA PROBLEME CUNOSCUTE

Această tehnică de abordare presupune cunoaşterea şi aplicarea de soluţii simple rezolvarea problemelor. Problemele mai complexe se pot descompune în sub-probleme sau se pot transforma în aşa fel încât să corespundă unei probleme simple.

Exemplul 1.

Să se determine într-un tablou toate numerele care se repetă.

Rezolvarea 1.

Se compară secvenţial fiecare număr cu toate celelalte. Complexitatea în acest caz este $O(n^2)$.

Rezolvarea 2.

Se sortează tabloul şi apoi se determină dacă numerele vecine au aceeaşi valoare. Complexitatea este O(n log n) când se utilizează un algoritm de sortare corespunzător.

Exemplul 2.

Să se determine dacă oricare trei puncte dintr-un plan sunt situate pe aceeaşi dreaptă.

Rezolvarea 1.

Se vor încerca grupurile distincte de câte 3 puncte. Complexitatea în acest caz este $O(n^3)$.

Rezolvarea 2.

Se utilizează exemplul 1 de mai sus. Complexitatea este $O(n^2 \log n)$. Se calculează panta dreptelor generate de fiecare două puncte din mulțime, se ordonează tripletele (punct, pantă, punct) după pantă şi pentru fiecare două triplete consecutive cu pantă egală se verifică dacă au un punct comun.

ABORDAREA EGOISTĂ

Abordarea egoistă sau lacomă (greedy approach) presupune utilizarea unei anumite metode până la epuizare şi apoi, pentru restul problemei, utilizarea până la epuizare a unei noi metode.

Principiile care stau la baza metodei sunt:

- se alege totdeauna soluția care pare cea mai bună la un moment dat, şi
- alegerea optimului local poate conduce la o soluție globală optimă.

Exemplul clasic se referă la eliberarea unei sume de bani (la bancomat) în bancnote de diferite valori. Se eliberează bancnote de valoarea cea mai mare până când acest lucru nu mai este posibil, apoi se trece la valoarea imediat următoare până când suma se completează.

DIVIDE ŞI CUCEREŞTE

Metoda divide şi cucereşte este o metodă eficientă care se bazează pe câteva principii foarte simple. Când o problemă este prea complexă, ea va fi împărțită în sub-probleme de întindere mai mică, se rezolvă recursiv fiecare sub-problemă şi se combină rezultatele. Este o abordare utilizată adesea în problemele care implică operații cu matrice.

```
Proc Resolve ( P: Problem; R:Result  )
Declare   P1, P2: Problem
Declare   R1, R2: Result
If Dimension(P)<=1
Then
   R:= Direct_result // rezultat direct
Endif
Divide P in P1 and P2
Resolve ( P1, R1 )
Resolve ( P2, R2 )
Combine ( R1, R2, R )
EndProc
```

Fig. 46 Rezolvarea recursivă a problemelor de tip divide şi cucereşte

Această abordare este eficientă mai ales dacă împărţirea se face de fiecare dată în două părţi egale. Dificultatea în tipul acesta de algoritmi este de a afla cea mai bună cale de a DIVIDE în sub-probleme şi de a face rezultatele să se COMBINE. Nu întotdeauna cea mai uşoară metodă de divizare permite adoptarea unei proceduri acceptabile de combinare a soluţiilor parţiale într-o soluţie globală.

Exemplu

Problema Max-Min. Să se găsească simultan maximul şi minimul unei liste de *n* numere.

Soluţia banală: se găseşte maximul apoi se găseşte minimul şi se întoarce ca rezultat cuplul astfel obţinut.

Divide şi cucereşte: Până când rămân doar două elemente, se împarte lista în două sub-liste, şi se obţine minimul şi maximul din sub-liste prin apel recursiv.

```
Func MaxMin( A, p, r)
// găseşte max şi min în A[p,r]
If p = r Then Return (A[p], A[p])
If p = r-1
Then
   If A[p] ≤ A[r]
      Then Return (A[r], A[p])
      Else Return (A[p], A[r])
   EndIf
EndIf
q ← (p+r)/2    // împărţire în numere întregi
(M1,m1) ← MaxMin( A, p, q)
(M2,m2) ← MaxMin( A, q+1, r)
Return(max( M1, M2), min( m1, m2))
EndFunc
```

Fig. 47 Algoritmul Max-Min rezolvat prin metoda divide şi cucereşte

PROGRAMARE DINAMICĂ

Programarea dinamică este o metodă de rezolvare ce se aplică în cazurile în care se caută soluții optimale ale problemelor pornind de la soluțiile optimale ale sub-problemelor în care acestea au fost împărțite. De asemenea se aplică pentru cazurile de suprapunere a sub-problemelor: sub-problemele îşi împart sub-sub-probleme.

Ca principii de bază, acest tip de rezolvare împarte problema generală în probleme punctuale ce pot fi mai uşor tratate, combină soluțiile sub-problemelor şi evită rezolvarea sub-problemelor mai mult de o dată prin păstrarea soluțiilor anterior obținute.

Se aseamănă cu metoda divizării şi cuceririi prin împărțirea problemei şi rezolvarea recursivă dar soluția se obține pornind de la rezolvarea unei sub-probleme mici şi creşterea ariei de aplicare (bottom-up) şi nu pornind de la problema în ansamblu şi apoi particularizând (top-down). Metoda „divide şi cucereşte" **nu** este bună pentru probleme în care soluția generală de dimensiunea *n* se bazează pe *n* sub-soluții de dimensiunea *n*-1 şi nici pentru probleme cu sub-probleme care se suprapun.

Exemple de domenii în care programarea dinamică se pretează sunt înmulțirea unui lanț de matrice, aflarea celei mai lungi secvențe comune a două căi într-o rețea etc.

Exemplu

Înmulțirea eficientă unui lanț de matrice – considerăm:

$A = A_0 \times A_1 \times \cdots \times A_{n-1}$.

Fiecare matrice A_i are dimensiunile $l_i \times l_{i+1}$ unde $0 \le i < n$. Costul calculării produsului $A_i \times A_{i+1}$ este $l_i \times l_{i+1} \times l_{i+2}$ operații. Ordinea în care se efectuează înmulțierea matricelor într-un lanț va duce la valori diferite de operații. Astfel, considerând A_0, A_1, A_2 de dimensiuni 50×5 , 5×100 , 100×10 numărul de operații este:

- $(A_0 \times A_1) \times A_2$: $50 \times 5 \times 100 + 50 \times 100 \times 10 = 75000$;
- $A_0 \times (A_1 \times A_2)$: $5 \times 100 \times 10 + 50 \times 5 \times 10 = 7500$.

Se observă că prin simpla rearanjare a ordinii de efectuare a operațiilor se pot realiza economii importante la timpul de calcul.

7.6. BIBLIOGRAFIE

[108] Ahuja RK, Magnanti TL, Orlin JB: *Network flows: Theory, algorithms, and applications*; Prentice Hall, Englewood Cliffs, NJ, 1993.
[109] Brassard Bratley: *Algorithmique, conception et analyse*; Masson 1987.

[110] Brookshear J. Glenn: *Computer Science - An Overview* (sixth edition); (11.5 Complexity of Problems); Addison-Wesley, 2000.
[111] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L.: *Introduction à l'algorithmique*; Dunod, Paris 1996.
[112] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L.: *Introduction to Algorithms*, McGraw-Hill, New York 1990.
[113] Cormen T. H.; C. E. Leiserson; R.L. Rivest: *Introduction to Algorithms*; The MIT Press, Cambridge, Massachusetts, 1990. 5.
[114] Dewdney A.D.: *The New Turing Omnibus*; (Cap. 15: Time and Space Complexity); Computer Science Press, 1997.
[115] Edelsbrunner Herbert: *Algorithms in combinatorial geometry*; Springer 1985.
[116] Fox BL: *Data structures and computer science techniques in operations research*; Operations Research, Vol.26, Nr.5, Operations Research Society of America, September-October 1978.
[117] Gibbons Alan: *Algorithmic graph theory*; Cambridge University Press 1985.
[118] Aho Alfred V., Hopcroft John E., Ullman Jeffrey D.: *Data structures and algorithms*; Addison-Wesley 1983.
[119] Horowitz Ellis, Sahni Sartaj, Anderson Freed S.: *L'essentiel des structures de données en C*; Dunod, Paris 1993.
[120] Knuth Donald. E.: *The art of computer programming*; 3 volumes, Addison-Wesley 1969.
[121] Mehlhorn Kurt: *Data structures and algorithms, Volume I: Sorting and Searching*; Springer 1984.
[122] Mehlhorn Kurt: *Data structures and algorithms, Volume II: Graph Algorithms and NP-Completness*; Springer 1984.
[123] Mehlhorn Kurt: *Data structures and algorithms, Volume III: Multidimensional Searching and Computational Geometry*; Springer 1984.
[124] Preparata F. P., Shamos M. I.: *Computational Geometry*; Springer 1985.
[125] Reingold E.M., Nievergelt J., Deo N.: *Combinatorial algorithms, theory and practice*; Prentice Hall 1977.
[126] Standish Thomas A.: *Data Structures, Algorithms & Software Principles in C*;(Cap. 6: Introduction to Analysis of Algorithms) Addison-Wesley, 1995.
[127] Weiss Mark Allen: *Data Structures and Algorithm Analysis in C++*; The Benjamin Cummings Publishing Company 1994.
[128] Wilf Herbert S.: *Algorithms and Complexity*; Internet Edition, Summer, 1994.

Capitolul 8

SORTAREA ŞI CĂUTAREA

8.1. SORTARE

Necesitatea sortării apare în rezolvarea multor probleme, fie ca cerinţă de sine stătătoare fie ca o facilitate auxiliară. În orice problemă care manipulează un volum mare de date, sunt necesare mecanisme pentru regăsire a structurilor de date dorite. Funcţionarea unor astfel de mecanisme poate fi mai rapidă dacă datele sunt stocate într-o ordine prestabilită bazată pe criterii de ordonare.

Principial, sortarea este un proces în care se introduce ca **intrare** o secvenţă $A = \langle a_1, a_2, \ldots, a_n \rangle$ şi ca ieşire se obţine o secvenţă $B = \langle b_1, b_2, \ldots, b_n \rangle$ care este o permutare a lui A cu proprietatea că $b_1 \le b_2 \le \ldots \le b_n$ sau $b_1 \ge b_2 \ge \ldots \ge b_n$. Există nenumărate modele de sortare fiecare cu diverse variante, toate dezvoltate pe măsură ce teoria generală a algoritmilor a evoluat.

Pentru aranjarea datelor se poate utiliza un criteriu sau mai multe, procesele multicriteriale fiind bazate pe procedee dezvoltate din mecanismele de sortare după o singură regulă.

În cele ce urmează vom considera, pentru simplitate, că se utilizează un singur criteriu de sortare, lucru ce este mai uşor de înţeles şi urmărit.

SORTAREA CU „BULE”

Este o metodă recunoscută ca înceată, dar este deosebit de simplă. Sortarea cu bule este una dintre metodele naturale, uşor de înţeles. Un exemplu este prezentat în figura următoare.

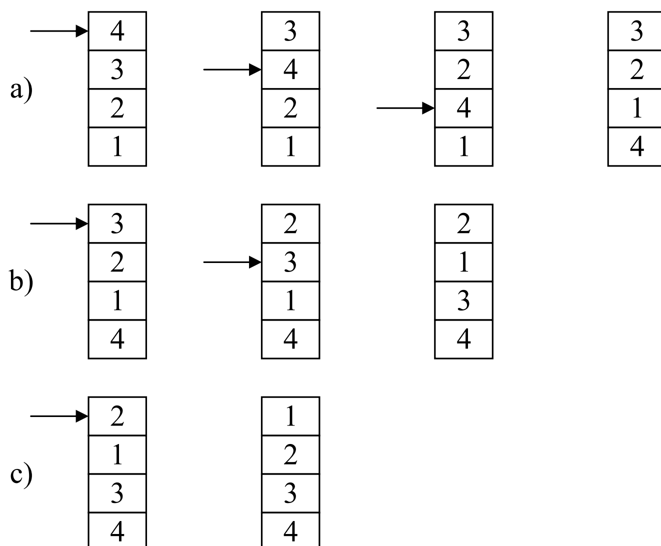


Fig. 48 Exemplu de sortare cu „bule”

Acest tip de sortare constă în parcurgerea vectorului în care sunt stocate datele şi compararea valorilor adiacente.

În cazul în care se constată că valorile sunt aranjate greşit se face schimbarea elementelor între ele. Operaţiile se reiau din vârful vectorul până când ordonarea este completă.

Considerând tabloul $A[1 \cdots N]$ care conţine valorile numerice ce trebuie ordonate ascendent se efectuează următoarele operaţii:

```
Repetă
   Parcurge A comparând valorile adiacente A[i], A[i+1]
   Dacă A[i] > A[i+1] atunci permută A[i] and A[i+1]
Până când nu mai există ce permuta
```

Fig. 49 Algoritmul conceptual pentru sortarea cu „bule”

La fiecare ciclu valorile cele mai mari apar la „suprafaţă” precum bulele de aer din apă, astfel încât sfârşitul tabloului este sortat (la fiecare ciclu o poziţie), iar valorile mai mici sunt deplasate în jos la fiecare ciclu o poziţie. Costul unui astfel de algoritm este $O(n^2)$.

Există mai multe versiuni de implementare, una dintre ele este prezentată în figura următoare:

```
For i← Dim(A) to 2
   For j← 1 to i-1
      If A[i] > A[i+1]
         A[i] ⇔ A[i+1]
      EndIf
   EndFor
```

Fig. 50 Algoritmul complet pentru sortarea cu „bule”

SORTAREA CU INSERARE

Sortarea cu inserare este una dintre cele mai simple metode. A fost imaginată din observarea comportamentului jucătorilor de cărți care o aplică instinctiv. Pentru acest tip de sortare se scoate un element şi se deplasează cele rămase. Se inserează elementul extras în poziția corectă. Aceste operații continuă până când procesul este complet. Atât cazul mediu cât şi cazul cel mai defavorabil $O(n^2)$ (Knuth 1998 [131]).

Evident pentru vârful vectorului nu avem ce verificare să facem de aceea se porneşte poziția a doua. Elementele de deasupra sunt deplasate în jos până când se găseşte poziția corectă pentru elementul extras. Se continuă cu elementul următor (poziția a treia). Procesul continuă prin inserarea fiecărui element la poziția corectă.

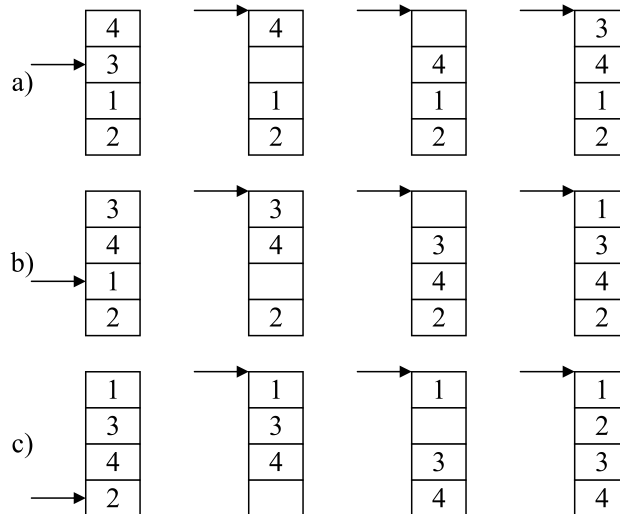


Fig. 51 Exemplu de sortare cu inserare

Dacă avem un vectorul cu dimensiunea *n*, utilizând sortarea cu inserare trebuie să parcurgem un număr de *n-1* elemente. Pentru fiecare, trebuie să examinăm şi să deplasăm până la n-1 alte elemente. Rezultă o complexitate a algoritmului de $O(n^2)$.

Avantajul acestui tip de sortare este că este un algoritm stabil şi nu necesită memorie suplimentară.

```
Parcurge vectorul element cu element
Pentru fiecare element
    Inserează elemental în poziţia corectă
    În vectorul cu elemente deja sortate
Până când ultimul element este introdus în poziţia corectă
```

Fig. 52 Algoritmul conceptual al sortării cu inserare

```
Proc Insert-Sort( A)
For i ← 2 to Dim(A)
   Tmp ← A[i]
   j ← i-1
   While j > 0 and A[j] > tmp
      A[j+1] ← A[j]
      j ← j-1
   EndWhile
   A[j+1] ← tmp
EndFor
```

Fig. 53 Algoritmul complet al sortări cu inserare

SORTAREA CU SELECŢIE

Este un algoritm elementar de sortare. Mecanismul său de lucru constă în găsirea celui mai mic element din vector şi aducerea sa pe prima poziţie.

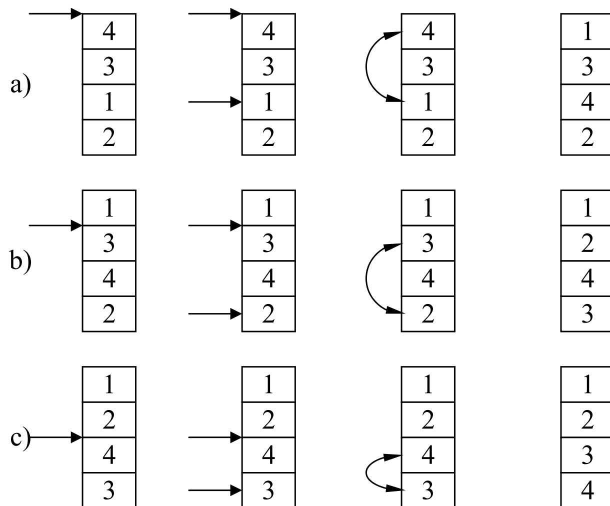


Fig. 54 Exemplu de sortare cu selecţie

Algoritmul conceptual al sortării cu selecție este prezentat în figura următoare:

```
Parcurge vectorul element cu element
Pentru vectorul neordonat
    Găseşte minimul
    Aduce minimul pe prima poziție
Până când vectorul este complet ordonat
```

Fig. 55 Algoritmul conceptual al sortării cu selecție

Pornind de algoritmul conceptual se poate scrie algoritmul complet pentru sortarea cu selecție.

```
Proc Select-Sort( A )

For i ← 1 To Dim(A)-1
   Min ← i
   For j ← i+1 to N
      If A[Min] > A[j]
      Then
         min ← j
      EndIf
   EndFor
   A[Min] ⇔ A[i]
EndFor

EndProc
```

Fig. 56 Algoritmul complet al sortării cu selecție

Presupunând că vectorul de intrare are dimensiunea *n*, complexitatea algoritmului este $O(n^2)$.

SORTAREA SHELL

Acest tip de sortare a fost dezvoltat de Donald L. Shell. Acest algoritm încearcă îmbunătățirea eficienței sortării cu inserare prin deplasarea rapidă a valorilor către destinația lor. Acest algoritm are complexitate $O(n^{7/6})$ în medie şi $O(n^{4/3})$ în cazul cel mai defavorabil (Knuth 1998 [131]).

Aceasta este prima încercare de algoritm de sortare care a reuşit să coboare sub complexitatea $O(n^2)$. Scopul este de compara şi dacă este necesar să schimbe poziția pentru elementele care sunt mai depărtate în tablou.

Considerăm exemplul din figura următoare:

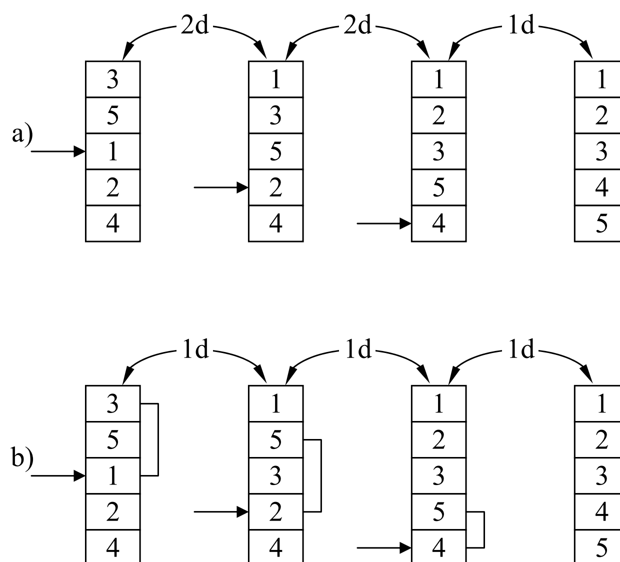


Fig. 57 Comparaţie între sortarea cu inserţie şi sortarea Shell

De fiecare dată când se schimbă între ele două elemente aflate la distanţa *k* se elimină un număr de k inversiuni. Aceasta permite înlocuirea a mai mult de o inversiune pe fiecare pas. Ideea este de a efectua sortarea cu inserare numai peste anumite elemente din tablou. Aceste elemente trebuie să fie alese cât mai depărtate unul de altul. În acest moment se vorbeşte de valoarea de incrementare. Pentru o valoare particulară a incrementului, de exemplu *h*, toate elementele situate la distanţa h se sortează utilizând algoritmul cu inserare. Aceasta conduce la câteva sub-tablouri. După fiecare sortare $A[i] \leq A[i+h]$ pentru toate valorile *i*.

Sortarea utilizează o serie de valori de incrementare care continuă atât timp cât incrementul este mai mare sau egal cu 1. Dacă secvenţa de incrementare este aleasă corespunzător viteza este deosebit de mare.

În cadrul metodei sortării cu inserţie (a) mai întâi extragem 1 şi deplasăm valorile 3 şi 5 cu o poziţie (în total două deplasări). În următoarea trecere sunt cerute tot două deplasări pentru a insera 2. În ultima evaluare se efectuează o singură deplasare. În total se fac 5 deplasări.

Simultan se prezintă şi sortarea Shell (b). Începem cu o distanţă de 2. Comparăm numerele 3 şi 1. Se extrage 1 şi se deplasează în sub-şir cu o poziţie în jos. Următoarea comparare este între 5 şi 2. Se extrage 2 şi se deplasează 5 cu o poziţie mai jos, inserând 2. după încheierea sortării cu intervalul 2 se trece la incrementul 1. În total vor fi 3 deplasări. Deci mai puţin decât sortarea cu inserare clasică.

```
Proc Shell_Sort( A, n)

Declare incr, I, j integer
Declare t

Incr ← n/2
While incr ≤ n
   I ← incr+1
   While i ≤ n
      t ← A[i]
      j ← i
      While j > incr
         If t<A[j-incr]
         Then A[j] ← A[j-incr]
         Else ExitWhile
         EndIf
         j ← j - incr
      EndWhile
      A[j] ← t
      i ← i + 1
   EndWhile
   incr ← incr/2
EndWhile
EndProc
```

Fig. 58 Algoritmul de sortare Shell

Secvența aceasta de $\frac{n}{2},\frac{n}{4},\ldots,1$ nu este cea mai rapidă soluție. Funcție de dimensiunea vectorului de intrare, se pot găsi secvențe de valori de incrementare care aduc viteza maximă.

SORTAREA RAPIDĂ (QUICKSORT)

Deşi sortarea Shell este semnificativ mai bună decât sortarea cu inserare se mai pot face îmbunătățiri. Una dintre metodele populare este sortarea „rapidă”.

Sortarea rapidă este un exemplu clasic de algoritm de tipul divide şi cucereşte.

Algoritmul de sortare funcționează prin partiționarea tabelului ce urmează a fi sortat şi apoi prin apelarea recursivă a sortării rapide. În procedura de partiționare se alege drept pivot unul dintre elementele tabelului. Toate valorile mai mici sunt deplasate în stânga şi toate elementele mai mari în dreapta.

În medie, complexitatea este $O(n \ln n)$.

Acest are forma:

```
Proc QuickSort (A:List; Lb, Ub: Integer)
if Lb < Ub
then
   M = Partition ( A, Lb, Ub )
   QuickSort ( A, Lb, M – 1 )
   QuickSort ( A, M, Ub )
EndIf
EndProc
```

Fig. 59 Algoritmul de sortare rapidă

Important este modul în care se realizează împărțirea listei în două sub liste. Pentru funcția care realizează acest lucru – *Partition* există mai multe formate. Unul dintre modurile de realizare este prezentat conceptual în figura mai jos.

```
Funcția Partajare

   Selectează un pivot din A[Lb]... A[Ub]
   Reodonează A[Lb]…A[Ub] aşa fel încât
       toate valorile din stânga pivotului sunt ≤ pivot
       toate valorile din dreapta pivotului sunt ≥ pivot
   Returnează poziția pivotului;
```

Fig. 60 Algoritmul conceptual pentru funcția de partajare

Pornind de la conceptul prezentat mai sus se poate concepe un exemplu de algoritm complet pe care în prezentăm în figura următoare.

```
Funct Partition (A:List;
                 Lb, Ub: integer):integer
   x ← A[Ub]
   i ← Lb – 1
   For j ← Lb To Ub-1
     If A[j]≤x Then
        i ← i+1
        A[i] ⇔ A[j]
     EndIf
   EndFor
   A[i+1] ⇔ A[Ub]
EndFunct
```

Fig. 61 Algoritm complet pentru funcția Partition

În algoritmul de mai sus semnul are ⇔ semnificația de permutare, de schimbare între ele a valorilor celor două variabile.

Un exemplu de sortare rapidă este prezentat schematic în figura de mai jos:

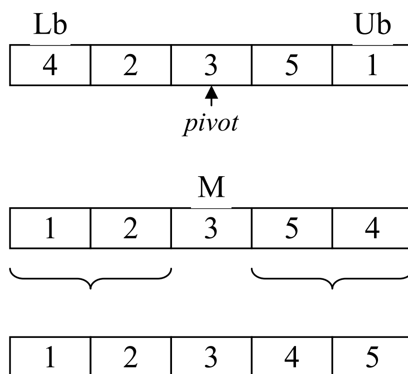


Fig. 62 Exemplu de sortare rapidă

Dezavantajele acestei metode sunt date de utilizarea de memorie suplimentară şi complexitatea în cazul cel mai defavorabil ($O(n^2)$). Totuşi cazul cel mai defavorabil este puțin probabil să se manifeste (Cormen 1990 [129]).

SORTAREA CU COMBINARE (MERGE SORT)

Acest tip de sortare se bazează pe împărțirea vectorului de intrare în două părți egale. Cele două părți se tratează recursiv prin sortare cu combinare după care se combină rezultatul sortării.

```
Proc MergeSort( E: List; first, last: integer )
Declare m Integer
If (first < last)
Then
   m ← (first+last)/2
   MergeSort( E, first, m )
   MergeSort( E, m+1, last )
   Merge( E, first, m, last )
EndIf
EndProc
```

Fig. 63 Algoritmul de sortare cu combinare

Importantă este partea combinare efectuată de procedura `merge()` care este apelată de procedura prezentată mai sus. Combinarea a două liste sortate de câte n/2 elemente necesită n-1 comparări în cel mai defavorabil caz. Acest lucru se face pornind de la începutul fiecărei liste şi eliminând în fiecare iterație cel mic element care se adaugă la lista finală.

Considerând L1 şi L2 liste de intrare şi L lista de ieşire, algoritmul conceptual este prezentat în figura următoare.

```
Procedura Combinare (L1, L2, L:Lista)
    // L1, L2 sunt liste de intrare
    // L Lista de ieşire

Cât timp L1 nu este vidă şi L2 nu este vidă
  Dacă primul element din L1 ≤ Dacă primul element din L2
    Atunci
       extrage primul element din L1 şi îl introduce în L
    Altfel
       extrage primul element din L2 şi îl introduce în L

Cât timp L1 nu este vidă
  Extrage primul element din L1 şi îl introduce în L
Cât timp L2 nu este vidă
  Extrage primul element din L2 şi îl introduce în L
```

Fig. 64 Algoritmul de combinare pentru sortarea cu combinare

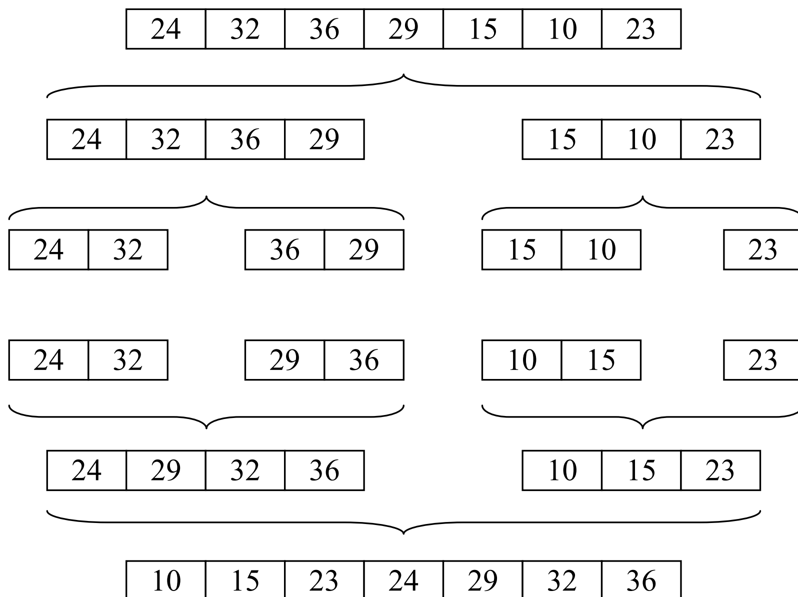


Fig. 65 Exemplu de sortare cu combinare

SORTAREA CU NUMĂRARE (COUNTING SORT)

Se presupune că toate cele n element sunt întregi şi cuprinse în intervalul de la 0 la k.

Pentru fiecare element de intrare se numără elementele mai mici. În final se plasează elementul chiar în poziția corectă.

Dacă vom considera lista de intrare ca fiind X şi lista de ieşire Y (nu se face sortarea direct în lista de intrare şi este nevoie de spaţiu suplimentar), atunci algoritmul are următoarea formă:

```
Proc counting_sort( x,y:List,n,k:Integer)
Declare z:List // zonă de lucru temporară

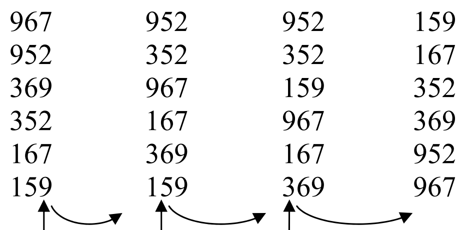
For i←0 to k-1 z[i] ← 0
For j←0 to n-1 z[x[j]] ← z[x[j]]+1
    // z[i] conţine numărul de elemente egale cu i
For i←1 to k-1 z[i] ← z[i] + z[i-1]
    // z[i] conţine numărul de elemente ≤i
For j←n-1 to 0
   y[z[x[j]]] ← x[j]; z[x[j]]← z[x[j]]-1;
EndFor
EndProc
```

Fig. 66 Algoritmul pentru sortarea cu numărare

Complexitatea algoritmului de sortare cu numărare este $O(n)$.

SORTAREA RADIX

Acest tip de sortare lucrează cel mai bine cu valori întregi şi presupune ordonarea valorilor funcţie de cifra cea mai puţin semnificativă, apoi funcţie de următoarea şi tot aşa până când se ajunge la cifra cea mai semnificativă.



967	952	952	159
952	352	352	167
369	967	159	352
352	167	967	369
167	369	167	952
159	159	369	967

Fig. 67 Exemplu de sortare radix

Sortarea radix se bazează pe baza de numeraţie (sau rădăcina ≡ radix) a reprezentării numerelor. Problema este dată de faptul că utilizează memorie suplimentară dar acest supliment este scăzut dacă se lucrează cu structuri dinamice.

Cheia este utilizarea unei tabele de dispersie cu dimensiunea 10 (pentru baza de numeraţie 10).

Complexitatea este de tipul $O(n)$ numărul exact de iteraţii depinzând de numărul maxim de cifre pe care îl au valorile ce se sortează.

SORTAREA CU GRĂMEZI (BUCKET SORT)

Se presupune că cele *n* valori de sortat sunt situate într-un interval oarecare. Se împarte intervalul în *M* sub-intervale egale. Se creează *M* grămezi fiecare corespunzând respectiv câte unui sub-interval. Se consideră pe rând valorile de sortat şi se introduc în grămada corespunzătoare. Se ordonează fiecare grămadă prin sortarea cu inserare.

În final se concatenează în ordine grămezile pentru a obţine rezultatul dorit.

Considerând un şir de *m* valori $A[i]$ cu proprietatea că $0 \leq A[i] < 1$ şi că s-au creat *n* grămezi $G[j]$ unde $j = 0,...,n-1$ un model algoritm de sortare cu grămezi poate fi cel din exemplul următor:

```
Funct BucketSort (A:List)
   Pentru I ← 1 la m
     Insereaza A[i] in G[j] unde
               este parte întreagă din n*A[i]
   Pentru j ← 0 la n-1 Sortare G[j]
   Concatenare G[0]...G[n-1]
EndFunct
```

Fig. 68 Algoritm pentru sortarea cu grămezi

În medie complexitatea este $O(n)$.

8.2. CĂUTAREA

CĂUTAREA DATELOR NEORDONATE

Pentru date neordonate nu prezintă un interes special. Ea se rezumă, pur şi simplu, la parcurgerea secvenţială a listei până când se întâlneşte valoarea căutată.

1	3	Lb
2	10	
3	20	
4	25	
5	29	
6	40	
7	41	
8	55	Ub

Fig. 69 Exemplu de listă

Presupunând că datele se află stocate într-un vector nesortat de dimensiune 8 numărul maxim de comparaţii pe care le poate face căutarea secvenţială este 8, deci

complexitatea este $O(n)$ în cel mai defavorabil caz. Făcând prezumția unei distribuții perfect aleatoare în medie numărul de paşi va fi n deci complexitatea este tot $O(n)$.

În continuare prezentăm algoritmul pentru căutare secvențială.

```
Funct SequentialSearch ( A: array;
                         Lb, Ub: integer;
                         key: Integer): integer;
   For I ← Lu to Ub do
      If A[i] = Key then
         Return i
      EndIf
   EndFor
   Return -1
EndFunct
```

Fig. 70 Algoritmul pentru căutarea secvențială

Evident căutarea secvențială este ineficientă şi este dovada faptului că datele nu trebuie păstrate niciodată fără să fie ordonate.

CAUTARE BINARĂ

Dacă există certitudinea că valorile au fost ordonate se poate utiliza căutarea binară. Acest tip de căutare implică alegerea elementului median şi împărțirea restului de elemente în două: submulțimea celor mai mici şi submulțimea celor mai mari. Dacă valoarea căutată este egală cu elementul median considerat ca pivot căutarea s-a încheiat. Dacă valoarea este mai mică respectiv mai mare atunci limitele mulțimii de căutare se ajustează la limitele submulțimii corespunzătoare. Operația se reia până când se găseşte valoarea sau mulțimea elementelor rămase este vidă.

```
Funct BinarySearch ( A: array;
                     Lb, Ub: integer;
                     key: Integer): integer
Repeat
   M ← (Lb*Ub)/2
   If key<A[M] then Ub ← M-1
   Else If key>A[M] then Lb ← M+1
   Else
     Return M
   EndIf
Until Lb > Ub
Return -1
EndFunct
```

Fig. 71 Algoritmul pentru căutarea binară (varianta iterativă)

Algoritmul poate fi prezentat în forma iterativă ca în figura de mai sus sau în forma recursivă prezentată în figura următoare.

```
Funct BinarySearchR ( A: array; Lb, Ub: integer;
                      key: Integer): integer
   If Lb > Ub Then
     Return -1
   EndIf
   M ← (Lb*Ub)/2
   If key<A[M] then Return BinarySearchR (A,Lb,M-1,Key)
   Else If key>A[M] then Return BinarySearchR (A,M+1,Ub,Key)
   Else Return M
   EndIf
EndFunct
```

Fig. 72 Algoritmul pentru căutarea binară (varianta recursivă)

8.3. BIBLIOGRAFIE

[129] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L.: *Introduction to Algorithms*, McGraw-Hill, New York 1990.

[130] Knuth Donald. E.: *The Art of Computer Programming*, Volume 1, *Fundamental Algorithms*. Addison-Wesley, Reading, Massachusetts 1968

[131] Knuth Donald. E.: *The Art of Computer Programming*, Volume 3, *Sorting and Searching*. Addison-Wesley, Reading, Massachusetts 1973, 1998.

[132] Niemann Thomas: *A Compact Guide to Sorting and Searching*; ePaper Press. 2003.

[133] Sedgewick R.: *Algorithms*; Addison-Wesley, Reading, Massachusetts, 1983, 2nd ed. 1988.

[134] Sedgewick, R.: *Communications of the ACM*, vol. 21, pp.847–857, 1978.

Capitolul 9

GRAFURI ŞI PROBLEME PE GRAF

9.1. ELEMENTE DE TEORIA GRAFURILOR

Istoric vorbind, teoria grafurilor începe în 1736 când Euler prezintă o teorie generală în care include şi rezolvarea „problemei podurilor din Königsberg”. Ulterior această teorie s-a dezvoltat din considerente practice. Nevoia de a stabili metode de minimizare a costurilor mijloacelor de transport, de comunicaţie, de aflare a drumului critic (acestea fiind doar câteva exemple) au impus implicarea teoreticienilor.

DEFINIŢII

Graful este un obiect matematic care este format din tripletul (N, A, f) unde N este un set de puncte (vârfuri sau noduri) şi A este un set de muchii (laturi sau arce), iar f este o funcţie de conexiune $f : A \mapsto N \times N$. Funcţia f stabileşte corespondenţa dintre fiecare arc punctele sale terminale din care cauză mai este numită ***funcţia arc–terminaţii*** (*edge-endpoint function*). A este un sub-multiset din $N \times N$, aceasta deoarece pot exista mai multe arce care să conecteze o pereche dată de noduri. În general, funcţia f se consideră implicit şi graful se notează $G = (N, A)$.

Numărul de noduri ale unui graf, notat $|G|$, reprezintă ***ordinul*** grafului. Numărul de muchii se notează $\|G\|$. Funcţie de ordinul său, un graf poate fi finit sau infinit. Dacă într-o problemă nu se specifică altfel atunci se consideră că graful este finit. Graful de ordinul 0 se numeşte ***graf vid***. Graful de ordinul 0 sau 1 se numeşte ***graf trivial***.

Conexiunile descrise prin setul de arce *A* pot fi *orientate* sau *neorientate*.

Un ***graf*** (sau ***graf neorientat*** sau ***graf nedirecționat*** sau ***graf ordinar***) *G* constă într-un set *N* de noduri şi un set *A* de arce dispuse în aşa fel încât fiecărui arc $a \in A$ i se asociază o pereche neordonată de noduri *p* şi *q* unde $p \in N$ şi $q \in N$. Dacă există un arc unic *a* asociat cu nodurile *p* şi *q* vom scrie $a = (p, q)$ sau $a = (q, p)$. În practică se întâlnesc adesea grafurile neorientate cum ar fi rețeaua de drumuri județene.

Un ***graf direcționat*** (sau ***graf orientat*** sau ***digraf***) *G* constă într-un set de set *N* de noduri şi un set *A* de arce dispuse în aşa fel încât fiecărui arc $a \in A$ i se asociază o pereche ordonată de noduri *p* şi *q* unde $p \in N$ şi $q \in N$. Dacă pentru un arc *a* asociat cu nodurile *p* şi *q* vom scrie $a = (p, q)$ rezultă că arcul are direcția de la *p* la *q*. Pentru $a = (q, p)$ arcul *a* are direcția de la *q* la *p*.

Un exemplu de graf orientat este alimentarea cu apă într-un oraş, unde apa poate circula numai de la furnizor spre consumatori niciodată invers. De la consumatori se adună reziduurile prin rețeaua de canalizare, un alt exemplu de graf direcționat (cel puțin în teorie).

Deoarece setul *A* al arcelor este descris prin perechile corespunzătoare de noduri terminale, în practică, notația generală pentru un graf *G* (direcționat sau nedirecționat) care este alcătuit din setul de arce *A* pe setul de noduri *N* este $G = (N, A)$. De cele mai multe ori funcția *f* este omisă în reprezentare şi este tratată implicit, ea fiind o simplă relație. De asemenea, se consideră implicit că *A* este o mulțime finită, iar *N* este diferit de mulțimea vidă.

Două noduri *p* şi *q* unite de un arc *a* sunt noduri ***adiacente*** sau ***vecine***. Dacă toate nodurile sunt adiacente între ele atunci graful este ***complet***.

Un arc *a*, care uneşte o pereche de noduri *p* şi *q* este ***incident*** pe *p* şi *q*. Despre *p* şi *q* se spune că sunt ***incidente*** la arcul *a*. Nodurile *p* şi *q* sunt capetele muchiei *a*.

Din definițiile anterioare nu rezultă necesitatea ca *p* şi *q* care descriu arcul $a = (p, q)$ să fie distincte. Avem deci posibilitatea ca un nod să fie conectat la el însuşi printr-un arc. O astfel de construcție $a = (p, p)$ se numeşte ***buclă***.

De asemenea, nu este obligatoriu ca arcul, orientat sau neorientat, care conectează două noduri să fie unic. Dacă există mai multe arce similare care conectează aceleaşi noduri se spune că sunt ***arce paralele***.

Dacă un graf nu are bucle şi nici arce paralele se numeşte ***simplu***. Aici trebuie notat că arcele orientate ce conectează aceleaşi două noduri dar în sens invers nu sunt considerate paralele.

Prin definiţie:

$G \cup G' := (N \cup N', A \cup A')$ şi

$G \cap G' := (N \cap N', A \cap A')$.

Dacă $G \cap G' = \phi$ atunci G şi G' sunt disjuncte.

Dacă $N' \subseteq N$ şi $A' \subseteq A$ atunci G' este un sub-graf al lui G (sau G este un super-graf al lui G').

Dacă $G' \subseteq G$ şi G' conţine toate muchiile $pq \in A$ cu $p, q \in N'$ atunci G' este un sub-graf indus al lui G. Se spune că N' induce sau acoperă G' pe G ($G' =: G[N']$). Astfel, dacă $M \subseteq N$ este un set de noduri, atunci $G[M]$ reprezintă graful pe M ale cărui muchii sunt acele muchii din G cu ambele capete în M. Graful $G' \subseteq G$ este un ***sub-graf acoperitor*** al lui G dacă N' acoperă tot G ($N' = N$).

Graful ponderat este acel graf pentru a căror elemente se ataşează valori de pondere. Aceste valori pot fi ataşate fie la muchii fie la noduri. Pentru muchii se pot ataşa viteze, timpi de întârziere, fluxuri etc. Pentru noduri se pot ataşa de exemplu datele despre populaţia pentru oraşe.

GRADUL UNUI NOD

Considerăm $G = (N, A)$ un graf nevid. Mulţimea vecinilor unui nod $p \in N$ este notată cu $V_G(p)$ sau pe scurt $V(p)$. Dacă în locul unui singur nod considerăm o submulţime $M \subseteq N$, vecinii din $N \setminus M$ ai nodurilor din M se numesc vecinii lui M şi mulţimea lor se notează $V(M)$.

Considerând un nod p, numărul de arce care sunt incidente pe nod $|A(p)|$ dă ***gradul*** sau ***valenţa*** acelui nod. Dacă un nod nu este conectat de nici un arc atunci gradul său este 0. Spunem că acest nod este izolat.

Dacă toate nodurile din graf au acelaşi grad se spune că graful este ***regulat***.

Gradul total al unui graf este dat de suma gradelor tuturor nodurilor.

Într-un graf neorientat gradul grafului este de două ori numărul arcelor. Pentru un graf orientat gradul său este dat de suma arcelor care intră pe fiecare nod, asta înseamnă în total numărul arcelor.

Un graf în care toate perechile de noduri sunt adiacente se numeşte ***graf complet***.

Un graf $G=(N,A)$ în care mulţimea nodurilor sale poate fi împărţită în două submulţimi disjuncte în aşa fel încât nici un arc nu are ambele terminaţii în aceeaşi submulţime. Dacă numărul de noduri este *k* atunci numărul de arce este C_2^K.

Un graf bipartit care are numărul maxim posibil de arce se numeşte graf bipartit complet. Numărul total de arce este în acest caz , $n \times m$ unde n şi m reprezintă numărul de noduri din fiecare dintre submulţimi.

DEFINIŢII SUPLIMENTARE

Un ***traseu*** (*walk*) într-un graf $G=(N,A)$ este o secvenţă finită de noduri şi arce adiacente între un nod de pornire n_0 şi un nod de final n_k: $A_{0,n}=\{(n_i,n_{i+1})\in A\}, i:=0,...,k-1$. Traseul trivial constă dintr-un singur nod. Lungimea traseului într-un graf este dată de numărul de arce $L_T=k$.

O ***cale*** sau un drum (*path*) este un traseu care porneşte dintr-un nod de start n_0 şi se încheie în nodul final n_k şi nu conţine arce care se repetă. O cale simplă între n_0 şi n_k nu conţine noduri care se repetă. Deoarece o cale este un traseu, lungimea unei căi este dată de definiţia din paragraful anterior.

Un graf G se numeşte ***conectat*** dacă pentru oricare pereche de noduri n_p şi n_q există un traseu care porneşte din n_p şi se încheie în n_q. În caz contrar graful este neconectat. Într-un graf conectat există o cale între oricare două perechi de noduri. Dacă într-un graf conectat, o ***punte*** (engl. bridge) este un arc a cărui eliminare transformă graful într-un graf neconectat.

Un traseu se numeşte închis dacă primul şi ultimul nod din secvenţă sunt identici. Un traseu închis care conţine cel puţin trei arce şi nodurile sunt distincte, cu excepţia primului şi ultimului, se numeşte ***ciclu***.

Două noduri sunt ***puternic conectate*** sunt noduri pentru care există o cale pe graf de la fiecare către celălalt. Graful puternic conectat este acel graf în care oricare două noduri sunt puternic conectate.

9.2. REPREZENTAREA GRAFURILOR

REPREZENTAREA GRAFICĂ

Este o reprezentare utilă pentru raţionamentul mintal al oamenilor. Pe asemenea reprezentări se pot uşor observa legături reale sau potenţiale. Este un suport pentru euristică şi intuiţie. Faptul că este atât de uşor de înţeles face ca acest tip de reprezentare să fie cel mai popular.

Nodurile şi arcele sunt reprezentate în plan prin puncte şi linii. Dacă un arc este orientat se adaugă o săgeată în capătul care arată direcție

În total putem distinge patru tipuri de grafuri:

- Grafuri: nu există restricții legate de bucle şi arce paralele;
- Grafuri simple: nu sunt permise bucle şi arce paralele;
- Grafuri orientate: toate arcele sunt orientate şi nu există restricții legate de bucle şi arce paralele;
- Grafuri orientate simple: toate arcele sunt orientate şi nu sunt permise bucle şi arce paralele.

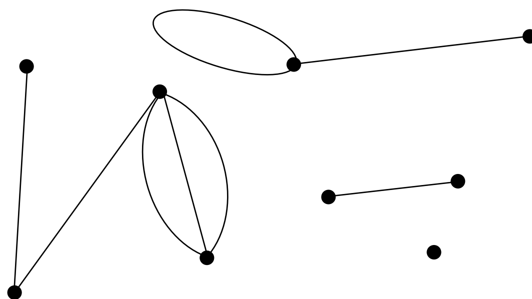


Fig. 73 Graf

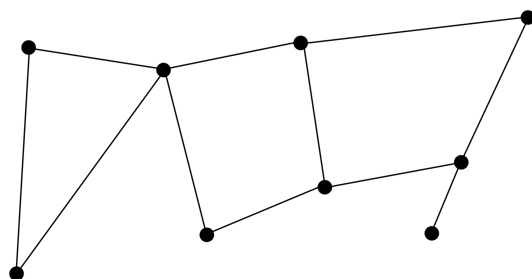


Fig. 74 Graf simplu

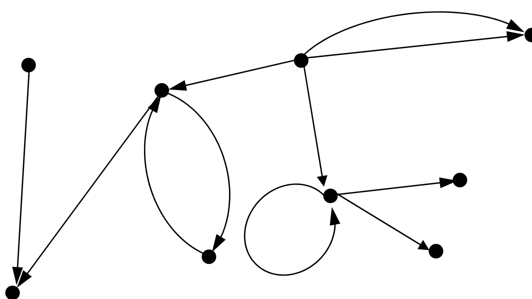


Fig. 75 Graf orientat

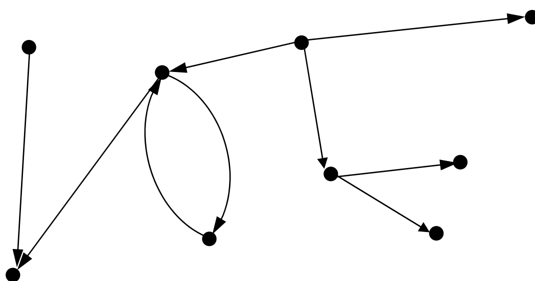


Fig. 76 Graf orientat simplu

Exemple

În viața cotidiană avem foarte multe cazuri concrete ce pot fi reprezentate prin grafuri.

Un astfel de exemplu de reprezentare prin grafuri poate fi conectarea calculatoarelor dintr-o instituție. În noduri sunt figurate calculatoarele, fie ele servere sau stații de lucru iar arcele reprezintă conexiunile cablate.

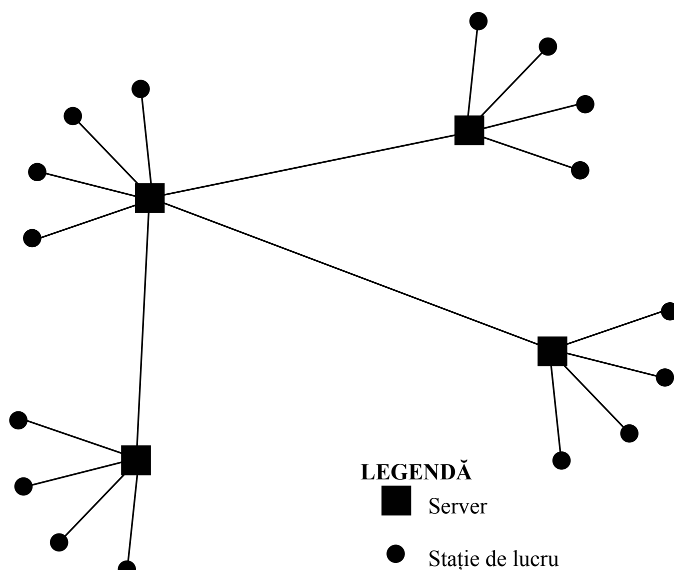


Fig. 77 Graf reprezentând conectarea calculatoarelor

Alt exemplu poate fi constituit de rețeaua de cale ferată ce uneşte localitățile dintr-o regiune. În noduri se situează localitățile iar arcele sunt liniile de cale ferată. Între unele noduri pot fi figurate arce paralele dacă linia este dublă sau triplă.

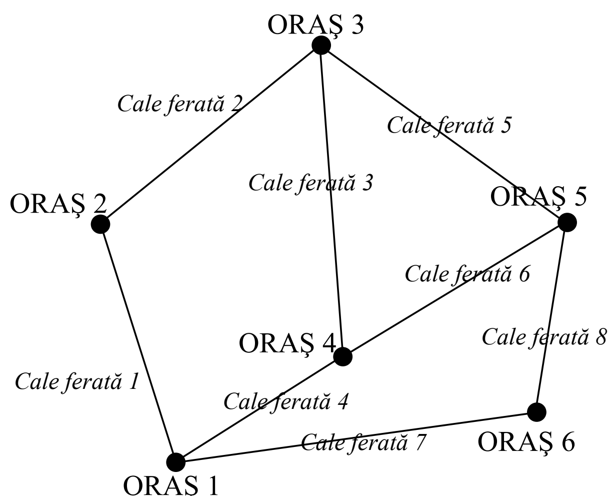


Fig. 78 Exemplu de graf reprezentând rețeaua de cale ferată dintr-o regiune

Reprezentarea utilizând grafurile nu este obligatoriu să se facă numai pentru situația când atât nodurile cât şi laturile sunt entități fizice. Se poate imagina un graf a persoanelor care se cunosc între ele.

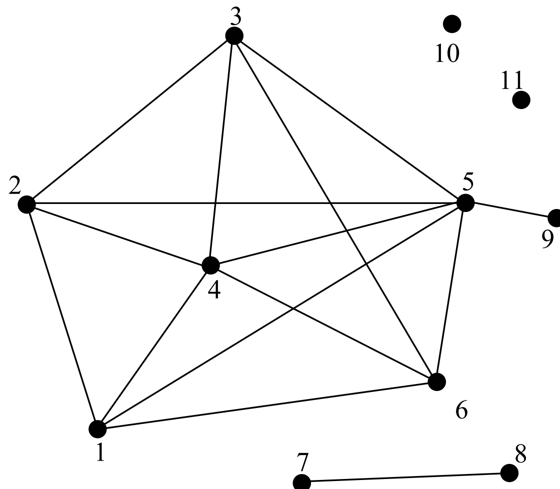


Fig. 79 Graful relațiilor dintre persoane

Graful prezentat în figura anterioară poate fi chiar orientat căci există cazuri frecvente când în relația dintre două persoane numai una cunoaşte pe cealaltă.

Pentru o rețea de drumuri în care fiecare segment are o anumită lungime există un anumit număr de maşini de intervenție situate în câteva localități importante. O astfel de situație este reprezentată în figura următoare.

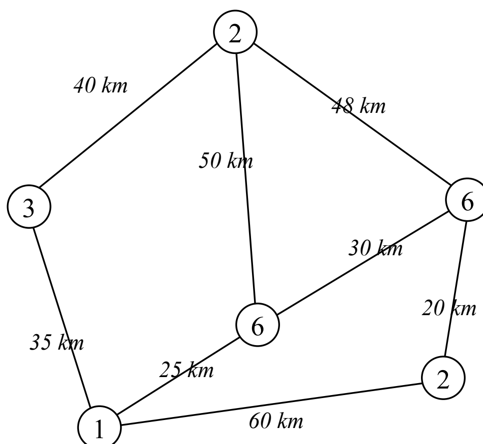


Fig. 80 Exemplu de graf ponderat

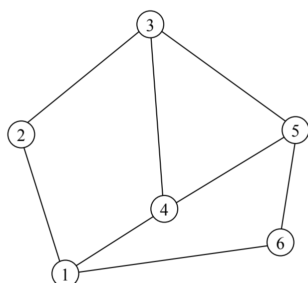
Numerele ataşate nodurilor reprezintă numărul de maşini care sunt disponibile în fiecare localitate. Muchiilor li se ataşează distanţa între localităţi.

REPREZENTAREA PRIN STRUCTURI DE DATE

Dacă reprezentarea grafică nu prezintă probleme deosebite, reprezentarea în memoria calculatorului nu mai este atât de simplă. Aceasta deoarece trebuie ţinut cont de spaţiul efectiv de stocare dar şi de timpul de prelucrare a datelor.

Există două căi principale de reprezentare a grafurilor: prin matricea adiacenţelor şi prin liste de adiacenţe.

Considerând un graf neorientat a cărui noduri sunt notate de la 1 la 6, reprezentarea sa poate fi văzută în figura următoare: a) forma matriceală şi b) lista de adiacenţă.



a)

	1	2	3	4	5	6
1	0	1	0	1	0	1
2	1	0	1	0	0	0
3	0	1	0	1	1	0
4	1	0	1	0	1	0
5	0	0	1	1	0	1
6	1	0	0	0	1	0

b)

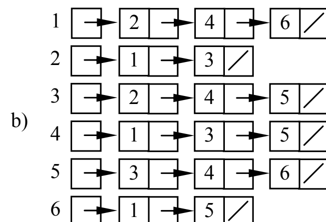


Fig. 81 Reprezentarea unui graf neorientat

Pentru reprezentarea prin matricea adiacenţelor în poziţia (*i,j*) s-a notat cu 1 faptul că nodurile *i* şi *j* sunt adiacente. Valoarea 0 indică inexistenţa unei muchii care să unească nodurile *i* şi *j*.

Similar putem reprezenta un graf orientat.

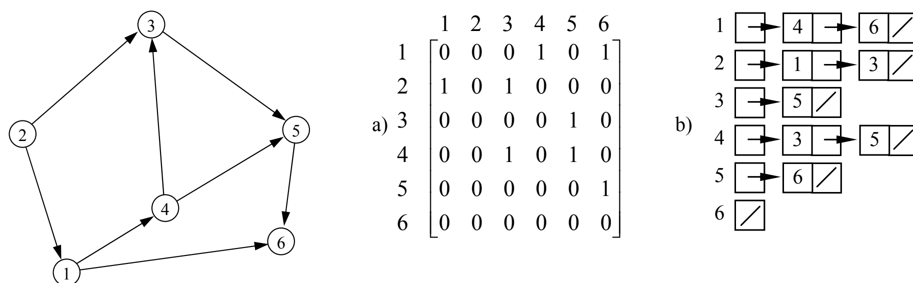


Fig. 82 Reprezentarea unui graf orientat

Din figura de mai sus, se observă imediat că spre deosebire de graful neorientat, reprezentarea matriceală a grafului orientat nu mai este obligatoriu simetrică.

Dacă graful este ponderat, atunci pe lângă indicaţia existenţei unei legături între două noduri se indică şi ponderea (sau distanţa, costul etc.) acelei legături. Acest lucru se poate face tot prin utilizarea matricei adiacenţelor şi a listei de adiacenţă. Lista de adiacenţă trebuie să conţină pe lângă numărul nodului şi valoarea care indică ponderea.

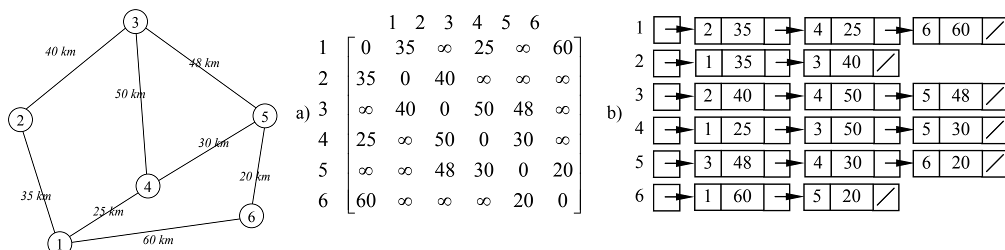


Fig. 83 Reprezentarea unui graf ponderat

Similar se poate reprezenta un graf ponderat orientat:

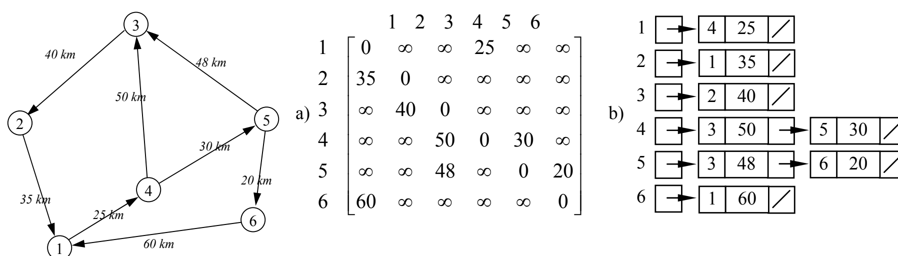


Fig. 84 Reprezentarea unui graf ponderat orientat

Şi în acest caz matricea adiacenţelor nu este obligatoriu simetrică.

În cazul reprezentării grafurilor trebuie să stabilim cantitatea de memorie necesară. Presupunând un număr de *n* noduri şi *m* arce rezultă:

- În cazul matricei adiacenţelor costul (în termeni de memorie) este n^2 ;
- Pentru liste de adiacenţă costul este $n+2m$ pentru grafuri neorientate şi $n+m$ pentru grafuri orientate

Deci reprezentarea cu ajutorul listelor de adiacenţă este mai eficientă decât utilizarea matricelor. De aceea este preferabilă folosirea listelor în reprezentarea, modelare şi rezolvarea problemelor în grafuri.

9.3. ARBORI

Un graf aciclic se numeşte pădure. O pădure conectată se numeşte arbore. Un nod al unui arbore a cărui grad este 1 se numeşte frunză. Un arbore netrivial are cel puţin 2 frunze.

Proprietăţi ale arborilor

Dacă un graf $G=(N,A)$ este arbore atunci:

- G nu are cicluri dar adăugarea unui arc între două noduri neadiacente va crea unul; G este aciclic maximal.
- Există o singură cale între oricare pereche de noduri;
- G este conectat dar eliminarea oricărui arc îl va transforma în graf neconectat; G conectat minimal.
- Dacă G are k noduri atunci el are exact $k-1$ arce.

ARBORELE DE ACOPERIRE

Arborele de acoperire pe un graf nedirecţionat $G=(N,A)$ este un subset de arce $T \subset A$ care nu conţine bucle, este aciclic şi conectează toate nodurile din N. Din condiţia de mai sus rezultă că dacă există n noduri arborele de acoperire va conţine exact $n-1$ arce.

Presupunem că există o funcţie a costurilor $c: A \to Z$, unde Z este mulţimea valorilor costurilor şi noţiunea de cost trebuie privită în sens larg, nu strict monetar. Costul total pentru arborele T este suma costurilor arcelor:

$$c(T)=\sum_{a \in T} c(a)$$

PROBLEMA ARBORELUI DE ACOPERIRE MINIMAL

Considerând o rețea (N, A), cu costurile c_{ij} asociate arcelor $(i, j) \in A$ să se găsească arborele de acoperire cu costul cel mai mic.

Exemplu: pe o rețea existentă (de drumuri sau de cale ferată) se pune problema de a se proiecta un nou sistem de comunicație (cu fibră optică etc.) care să conecteze un set de locații importante (cum ar fi districtele) la un cost minimal.

> ***Teoremă:*** *Fie U un subset corespunzător al setului de noduri N. Dacă (u, v) este arcul cu costul cel mai mic astfel încât $u \in U$ şi $v \in N \setminus U$ atunci există un arbore de acoperire minimal care include arcul (u, v).*

Demonstrație: Să presupunem că nu există nici un arbore de acoperire minimal care să includă arcul (u, v). Fie *T* un arbore de acoperire minimal oarecare. Adăugând (u, v) la *T* vom introduce un ciclu (*T* este arbore de acoperire). În consecință, există un alt arc (u', v') în *T* astfel încât $u' \in U$ *şi* $v' \in N \setminus U$. Dacă nu, ar fi imposibil să realizăm un ciclu între u şi v fără să parcurgem de două ori arcul (u, v).

Ştergând arcul (u', v') întrerupem ciclul şi obținem un arbore de acoperire T' a cărui cost nu este mai mare decât *T* de vreme ce am presupus din start că $c_{uv} \le c_{u'v'}$. Deci, T' contrazice presupunerea că nici un arbore de acoperire minimal nu include arcul (u, v).

Algoritmul generic de aflare a arborelui de acoperire minimal este dat de:

```
Proc AMA_Generic( G, cost )

T ← ∅
Cât timp T nu este arbore de acoperire
   Caută arc (u,v)∈G-T astfel încât
         u∈T şi cost(u,v) minimal
   T←T∪(u,v)
```

Fig. 85 Algoritmul generic pentru găsirea arborelui minimal de acoperire

Algoritmul lui Prim pentru arborele minimal de acoperire

În cadrul algoritmului lui Prim, *X* reprezintă un arbore unic şi S setul de noduri ale acestui arbore. În fiecare iterație se adaugă arborelui câte un nou nod până când toate nodurile sunt cuprinse.

Algoritmul conceptual este prezentat mai jos.

```
Proc AMA_Prim ( G, cost )

Alege un nod de start s
T ← ({s}, ∅)

Cât timp V[T] ≠ V
   Caută cel mai ieftin arc (u,v)
         care conectează T de G-T
         //la egalitate alege aleator
   T ← T∪({v},{(u,v)})
Returnează T
```

Fig. 86 Forma conceptuala a algoritmului lui Prim

Algoritmul lui Prim este de tipul $O(n^2)$.

Algoritmul lui Kruskal

Pentru rezolvarea problemei arborelui minim de acoperire algoritmul Kruskal utilizează o abordare diferită de algoritmul Prim. În loc să dezvolte un arbore unic, algoritmul Kruskal selecționează cele mai arcele cu costul cel mai mic dezvoltând câți arbori sunt necesari, arbori ce în final se vor uni în procesul de adăugare de noi arce.

Inițial fiecare nod din graf este considerat ca un arbore trivial.

```
Proc AMA_Kruskal ( G, cost )

T ← (N, ∅)
Ordonează arcele în ordine ne-descrescătoare

Pentru fiecare (u,v)∈A[G] (ordonate)
   Dacă u,v sunt în diferite
        componente conectate ale lui T
   T ← T ∪ {(u,v)})

Returnează T
```

Fig. 87 Forma conceptuala a algoritmului lui Kruskal

Algoritmul lui Kruskal are complexitatea $O(m \log m)$, unde m este numărul de arce.

Exemple de utilizare a algoritmilor de găsire a arborelui de acoperire minimal sunt prezentate mai jos. Se poate observa modul de acțiune şi deosebirile dintre cele două abordări. Considerăm un caz particular dat de un graf cu 9 noduri şi 12 arce cu costurile prezentate în figura următoare. În continuare prezentăm dezvoltarea arborelui de acoperire minimal prin algoritmul Prim şi prin algoritmul Kruskal .

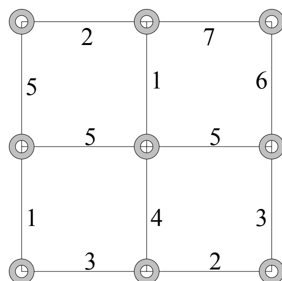


Fig. 88 Graful considerat pentru dezvoltarea arborelui de acoperire minimal

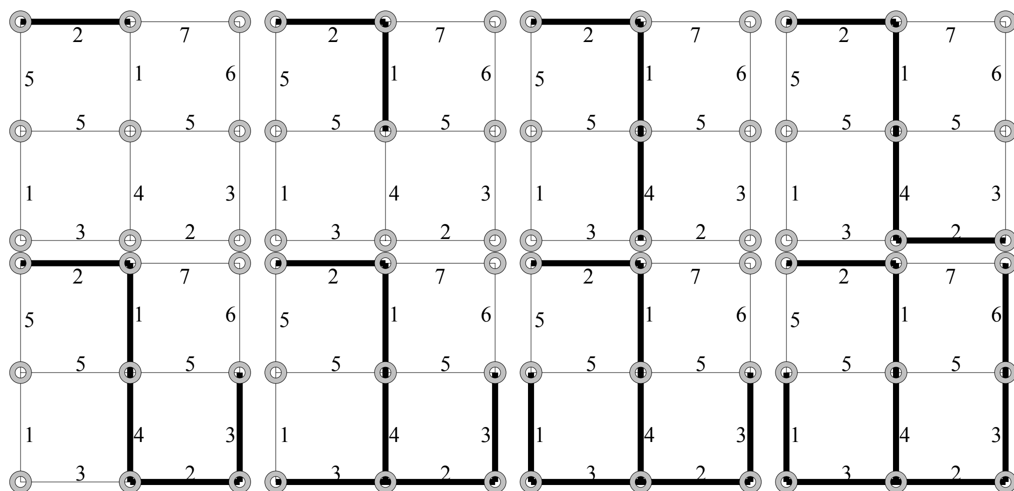


Fig. 89 Rezolvare cu algoritmul Prim

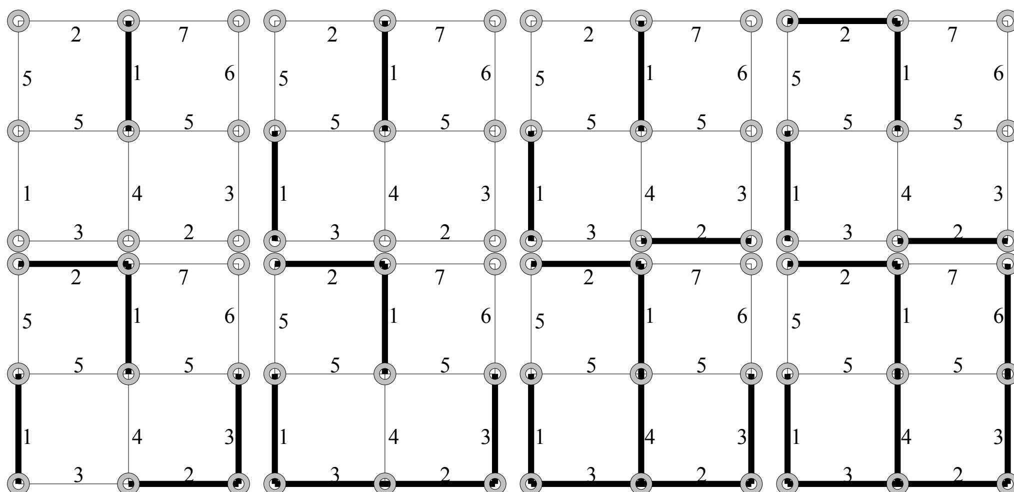


Fig. 90 Rezolvare cu algoritmul Kruskal

9.4. PARCURGEREA GRAFURILOR

Parcurgerea grafurilor trebuie făcută într-o manieră organizată, sistematică pentru a nu scăpa noduri sau arce. Există două tehnici importante:

- Căutare pe orizontală (breadth-first search): se caută să se obțină maximum de lărgime din examinarea unui nod şi apoi vizitarea imediată a nodurilor adiacente.
- Căutare pe adâncime (depth-first search): se caută să se obțină maximum de adâncime prin vizitarea unui nod şi apoi efectuare de căutare recursivă pe adâncime pentru fiecare nod adiacent.

9.5. PROBLEME DE ACOPERIRE PE REȚELE

PODURILE DIN KÖNIGSBERG

Königsberg, oraş din Prusia orientală (astăzi cunoscut sub numele de Kaliningrad în Rusia), este aşezat pe râul Pregel care înconjoară insula Kneiphof, după care se desparte în două brațe. Apa delimitează terenul în patru zone distincte. Malurile râului şi insula sunt legate prin şapte poduri.

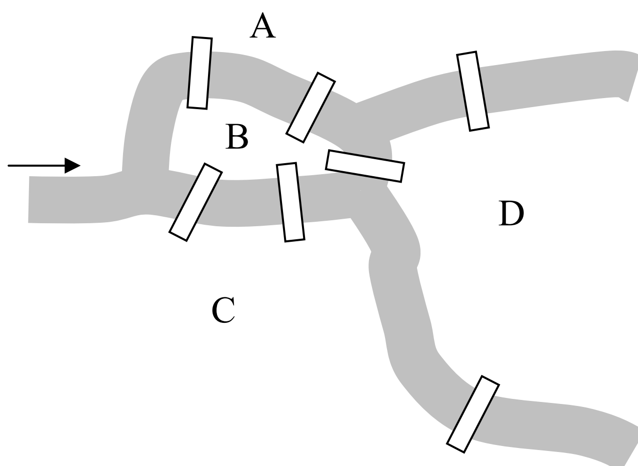


Fig. 91 Harta podurilor din Königsberg

La începutul secolului 18, orăşenii şi-au pus întrebarea: „Este posibil să facem o plimbare prin oraş trecând pe fiecare din cele şapte poduri o singură dată şi să ne în toarcem de unde am plecat?” Matematicianul elvețian Leonhard Euler rezolvă problema în 1731, fapt care constituie actul de naştere al teoriei grafurilor.

Logica ne spune că oricare punct din fiecare din cele patru zone este echivalent şi deci le putem trata unitar prin reducerea la unul singur (nodul). Podul

reprezintă calea de legătură (arcul). Problema poate fi deci reprezentată în mod abstract printr-un graf.

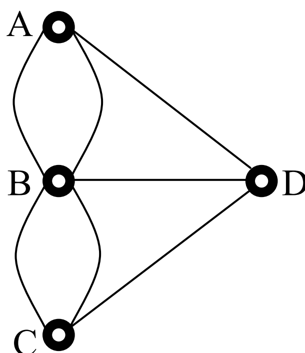


Fig. 92 Reprezentarea sub formă de graf

În conformitate cu teoria grafurilor problema se defineşte: *Să se găsească un traseu acoperitor pe rețea care porneşte şi se încheie în acelaşi nod parcurgând toate arcele* ***exact o singură dată****.*

Acea cale care acoperă graful, pornind dintr-un nod şi încheindu-se în acelaşi nod şi traversează fiecare arc exact o singură dată se numeşte parcurs eulerian. Graful care conține un parcurs Eulerian poartă numele de graf Eulerian.

> ***Teorema lui Euler*** *– un graf G este Eulerian dacă şi numai dacă toate nodurile au grad par.*

Fleury's Algorithm

Fie $G = (N, A)$ un graf conectat. Dacă G este Eulerian atunci următorul algoritm va produce un traseu Eulerian pe G.

Pas 1. *Alege oricare nod* ***v*** *din* ***G*** *şi fixează-l drept nod curent şi inițializează traseul cu o secvența de arce vidă.*
Pas 2. *Selectează oricare arc* ***a*** *incident din nodul curent, dar nu se alege o punte decât dacă nu există alternative.*
Pas 3. *Adaugă* ***a*** *la secvența de arce care constituie calea şi stabileşte celălalt capăt al arcului* ***a*** *ca nod curent.*
Pas 4*. Elimină* ***a*** *din graf. Elimină toate nodurile izolate.*

***Pas 5**. Dacă **G** este vid atunci STOP. Altfel trece la Pas 2.*

PROBLEMA POŞTAŞULUI CHINEZ

Problema poştaşului chinez este legată de eficienţa deplasării unui poştaş pentru livrarea corespondenţei într-un mediu urban. El trebuie să străbată cel puţin o dată toate străzile pornind dintr-un nod, unde se presupune că se află oficiul poştal, şi să se întoarcă în punctul de pornire. Distanţa total parcursă trebuie să fie minimă.

Problema a fost prima dată studiată de matematicianul chinez Meigo Guan în anul 1962, de unde provine şi numele.

Din punctul de vedere al teoriei grafurilor problema are formularea:

Să se găsească un circuit care traversează fiecare arc a unui graf $G = (N, A)$ *cel puţin o dată şi pentru care suma costurilor este minimă.*

Găsirea soluţiei optimale într-un graf orientat sau neorientat este NP-completă.

Chiar dacă a intrat în istoria teoriei grafurilor sub denumirea de „problema poştaşului” o aplicaţie esenţial este şi aceea a inspecţiei drumurilor dintr-o reţea (presupunem că pentru determinarea stării tehnice). De aceea problema mai poate fi întâlnită în literatură şi sub denumirea de problema inspecţiei drumului (engl. route inspection problem).

CICLU HAMILTONIAN

Considerând un graf neorientat $G = (N, A)$ să se găsească un traseu care vizitează fiecare nod exact o singură dată şi se întoarce în nodul de pornire.

Dacă un asemenea traseu există un asemenea traseu atunci graful se numeşte graf Hamiltonian iar traseul ciclu Hamiltonian.

Deşi par să existe similarităţi cu graful Eulerian există mari diferenţe.

Intuitiv s-ar spune că este cu atât mai probabil ca un graf să fie Hamiltonian cu cât există mai multe arce. Dirac, în 1952, a demonstrat teorema care îi poartă numele:

Oricare graf nu un număr de noduri $n \geq 3$ *şi gradul minim al nodurilor cel puţin egal cu* $n/2$ *are un ciclu hamiltonian.*

Utilizând această teoremă putem afla dacă avem sau nu un graf Hamiltonian. Aflarea efectivă a circuitului este însă ceva mai dificilă pentru că nu există algoritmi care să rezolve problema într-un timp rezonabil.

Problema este NP-completă.

PROBLEMA VÂNZĂTORULUI AMBULANT

Există multe aplicații care au legătură cu ciclul Hamiltonian. Între acestea este problema vânzătorului ambulant.

Această problemă are forma: pe o rețea în care nodurile reprezintă oraşele, un vânzător trebuie să viziteze toate oraşele cel puțin o dată cu costuri minime, cunoscut fiind costul călătoriei pe fiecare latură a rețelei.

Problema se simplifică dacă graful considerat este Hamiltonian. Vânzătorul va trece prin fiecare nod exact o singură dată. În acest caz algoritmul poate fi constituit din următorii paşi:

Pas 1. *Listează toate ciclurile Hamiltoniene.*
Pas 2. *Calculează costurile pentru fiecare ciclu*
Pas 3. *Alege ciclul cu costul cel mai mic.*

Trebuie notat că pot exista mai multe cicluri cu acelaşi cost.

Această formă a problemei are cea mai bună soluție de complexitate exponențială.

Este o problemă de optimizare discretă sau combinatorică şi un exemplu arhicunoscut de ilustrare a clasei de probleme din teoria complexității computaționale greu de rezolvat. Dacă nu există un ciclu Hamiltonian problema este de complexitate factorială (*n!*).

9.6. BIBLIOGRAFIE

[135] *** – *Duden: Informatik*; Dudenverlag, Mannheim a.o., 1993.
[136] Aho A., J. E. Hopcroft, J. D. Ullman: *Data Structures and Algorithms*; Addison-Wesley Publishing Company, Reading Massachusetts, 1983 .
[137] Brandstädt: *Graphen und Algorithmen*; Teubner, Stuttgart, 1994.
[138] Chartand G., O. R. Oellermann: *Applied and algorithmic graph theory*; McGraw-Hill, New York, 1993.
[139] Cormen T. H., C. E. Leiserson, R.L. Rivest: *Introduction to Algorithms*; The MIT Press, Cambridge, Massachusetts, 1990.
[140] Diestel Reinhelt: *Graph Theory*; Electronic Edition 2000, Springer-Verlag, New York 2000.
[141] Fields J. E.: *Introduction to Graph Theory*; Southern Connecticut State University, Department of Mathematics, 2001.
[142] Preparata F. P.; M. I. Shamos, Computational Geometry, Springer, New York, 1988.
[143] Sedgewick R.: *Algorithms*; Addison-Wesley, Reading, Massachusetts, 1983, 2nd ed. 1988.

Capitolul 10

PROBLEME DE MINIM ŞI MAXIM PE REŢEA

10.1. INTRODUCERE

Să considerăm un graf orientat simplu $G = (N, A)$ în care N este mulţimea nodurilor şi A este mulţimea arcelor unde N are dimensiunea n şi A are dimensiunea m. Fie $c : A \to R$ o funcţie care ataşează un cost c_{ij} fiecărui arc $(i, j) \in A$. Date fiind două noduri speciale o şi d numite origine şi destinaţie să se găsească o cale între o şi d în aşa fel încât costul total (suma costurilor arcelor incluse) să fie minim.

Corolarul acestei probleme este să se găsească drumul cel mai lung între origine şi destinaţie. Se mai numeşte şi „problema drumului critic”.

Pe lângă calea cea mai scurtă şi cea mai lungă este de multe ori necesar să se listeze în ordine cele mai scurte sau cele mai lungi k drumuri între origine şi destinaţie.

Toate aceste probleme au numeroase aplicaţii în industria transporturilor şi în administrarea infrastructurii transporturilor. Majoritatea programelor de optimizare din acest domeniu utilizează algoritmi de aflare a drumului (drumurilor) minim (minime).

În continuare vom prezenta metode de găsire a căii minime între origine şi destinaţie şi problema primelor k drumuri minime între origine şi destinaţie insistând asupra celei de-a doua deoarece este mai generală şi cu aplicaţii multiple în transporturi, managementul infrastructurii, logostică etc.

10.2. DRUMUL CEL MAI SCURT

Aflarea drumului cel mai scurt este o problemă studiată extensiv în anii 1950. Este o problemă relativ uşoară pentru o rețea cu n noduri şi m arce rezolvarea poate fi aflată cu un algoritm de complexitate $O(n^2)$. Chiar dacă această complexitate a fost demonstrată şi implementată în numeroase programe practice, totuşi se mai lucrează la îmbunătățirea şi simplificare algoritmului.

Diferite îmbunătățiri au fost aduse şi complexitatea este prezentată în tabelul următor.

Tabelul 9. Evoluția complexității algoritmilor de aflare a drumului minim

Algoritm	Complexitate
Dijkstra – 1959	$O(n^2)$
Williams – 1964	$O(m\log_2 n)$
Johnson – 1977	$O(m\log_d n), d = \max(2, \frac{m}{n})$
Boas et al. – 1977	$O(c + m\log_2 c\log_2 \log_2 c)$
Johnson – 1982	$O(m\log_2 \log_2 c)$
Fredman&Tarjan – 1981	$O(m + n\log_2 n)$
Gabow – 1985	$O(m\log_d c), d = \max(2, \frac{m}{n})$
Tarjan et al. – 1989	$O(m + n\sqrt{\log_2 c})$

Această problemă are câteva abordări standard:

Sursă unică – destinație unică: considerând o origine trebuie să se găsească drumul cel mai scurt până la un nod numit destinație;

Sursă unică – toate destinațiile: considerând un nod de origine trebuie găsite drumurile cele mai scurte către toate celelalte noduri din rețea.

Toate perechile: trebuie găsite drumurile minimale între toate perechile de noduri. Această problemă poate fi văzută şi ca n probleme de sursă unică şi rezolvată ca atare.

Problema căii minime este una dintre cele mai studiate probleme în cercetările operaționale privind optimizarea traficului în rețele. În domeniul transporturilor şi al administrării infrastructurii transporturilor se utilizează atât problema simplificată a determinării drumului minim (din punctul de vedere al distanței, timpului, costului etc.) între o origine şi o destinație dar şi alte variante de acțiune sub anumite restricții suplimentare sau structuri de graf particulare.

Datorită naturii aplicaţiilor, în alegerea algoritmilor specifici specialistul în ingineria transporturilor trebuie să aibă o abordare flexibilă atât din punctul de vedere al timpului de rulare cât şi din punctul de vedere al memoriei utilizate. Deoarece nu există un algoritm pe care putem să-l considerăm cel mai bun, alegerea se va baza pe rezolvarea condiţiilor considerate. Datorită dimensiunii deosebite a problemei, o atenţie deosebită trebuie alocată alegerii şi implementării unor structuri de date eficiente (Tarjan et al. 1983 [175]).

ABORDĂRI ALE PROBLEMEI DRUMULUI CEL MAI SCURT ÎN CĂI DE COMUNICAŢIE

Analiza şi planificarea transporturilor au utilizat în permanenţă proceduri de calcul al drumului cel mai scurt. Modelând reţele de mari dimensiuni, cu multe condiţii şi având permanent nevoie de a compara variante multiple, specialiştii în cercetări operaţionale au avut nevoie de algoritmii cu permanenţe mereu îmbunătăţite. Performanţele mai bune au condus la modele mai bune care la rândul lor au necesitat inventarea de algoritmi mai performanţi.

Studiind problema, Pallottino şi Scutellà de Universitatea din Pisa au găsit că după 1950 pe plan mondial s-au publicat peste două mii de lucrări ştiinţifice în reviste de prestigiu şi convolute de congrese internaţionale (Pallottino şi Scutellà 1997 [165]). Numărul real al articolelor este cu siguranţă mult mai mare. Ei au găsit aceste lucrări atât în publicaţii din domeniul teoriei grafurilor cât şi în reviste de alte specializări: transporturi, calculatoare, telecomunicaţii, management, cercetărilor operaţionale etc. Aceasta denotă interesul deosebit acordat problemei şi importanţa ca soluţia să fie una eficientă din punctul de vedere al costurilor. De aceea există numeroase variante de algoritmi, unele fiind îmbunătăţiri ale variantelor anterioare, numeroşi fiind autorii care şi-au adus contribuţie. Pentru treceri în revistă asupra subiectului a se vedea: (Johnson 1973(a) [163], Denardo şi Fox 1979 [152], [153], Shier şi Witzgall 1981 [171], Deo şi Pang 1984 [154], Gallo şi Pallottino 1986 [161], Bertsekas 1991b [149], Ahuja, Magnanti şi Orlin 1993 [144], Cherkassky, Goldberg şi Radzik 1996 [151].

Majoritatea sistemelor software utilizate în planificarea transporturilor posedă module de calcul a căii minime. Un interes deosebit se manifestă şi în domeniul planificării lucrărilor de construcţie unde distanţele până la sursele de aprovizionare şi costurile asociate sunt elemente critice. Pentru treceri în revistă asupra algoritmilor căii minime în transporturi a se vedea: Steenbrink 1974 [174], Van Vliet 1978 [176], Gallo şi Pallottino 1984 [159], Pallottino şi Scutellà 1997 [165].

ALGORITMUL LUI DIJKSTRA

Acest algoritm a primit numele inventatorului său, olandezul Edsger Dijkstra. Rezolvă problema drumului minim pentru un graf $G(N, A)$ conectat şi orientat a cărui costuri pe arce sunt ne-negative. Primeşte ca date de intrare un graf G cu costuri

ne-negative şi un nod de pornire – originea. Algoritmul are o abordare egoistă. La fiecare iteraţie se caută nodul cel mai apropiat de origine.

Ideea care stă la baza algoritmului este faptul că orice drum minimal este alcătuit din segmente care la rândul lor sunt minimale.

Presupunem că funcţia $c: N \times N \rightarrow [0, \infty]$ descrie costurile $c(u,v)$ de deplasare de la nodul u la nodul v pe arcul corespunzător – pentru nodurile neconetate presupunem că distanţa este infinit. Costul drumului dintre două noduri este suma arcelor ce constituie drumul.

Algoritmul Dijkstra funcţionează prin construirea unui sub-graf *S* al lui *G* în aşa fel încât distanţa de la origine la orice alt nod este minimă. Iniţial *S* conţine doar originea cu distanţa 0.

La fiecare iteraţie se identifică nodul a cărui distanţă până la origine este minimă şi se adaugă la arbore. Algoritmul se încheie când *S* devine un arbore de acoperire pe *G* sau când toate nodurile de interes au fost atinse.

```
Algoritmul Dijkstra

Iniţializează: S ← {s}; d[s] ← 0;
Iniţializează: d(v) ← c(s,v) pnrtu v∈N(G)-S

Cât timp |S|≠|N|
   Alege u din N-S încât d(u) este minim
   S ← S ∪ {u}
   Pentru fiecare nod v ∈ N-S caculează
       d(v) ← min(  d(v), d(u)+c(u,v) )
```

Fig. 93 Algoritmul conceptual Dijkstra

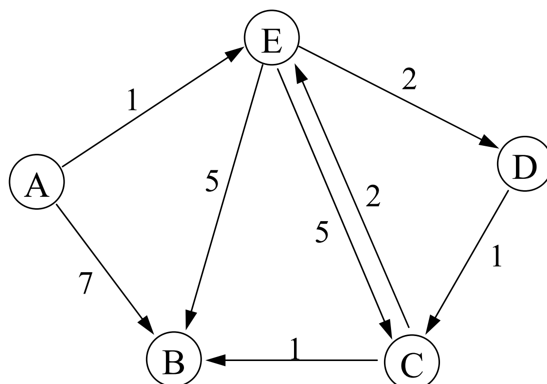


Fig. 94 Graful pentru tratare cu algoritmul lui Dijkstra

	S	u	d[B]	d[C]	d[D]	d[E]
0	{A}	-	7	∞	∞	1
1	{A,E}	E	6	6	3	1
2	{A,E,D}	D	6	4	3	1
3	{A,E,D,C}	C	5	4	3	1
4	{A,B,C,D,E}	B	5	4	3	1

Fig. 95 Exemplu de rezolvare cu algoritmul lui Dijkstra

Algoritmul lui Dijkstra este de tipul stabilirea etichetei deoarece, de fiecare dată când un nou nod este ales să facă parte din *S*, distanţa din origine şi traseul fixat sunt definitive.

Acest algoritm are complexitatea de rulare $O(n^2)$.

ALGORITMUL BELLMAN-FORD

Fie $C_i(t)$ lungimea celei mai scurte căi nodul *i* şi destinaţie (nodul *n*) care este alcătuit din cel mult *t* arce. Considerăm $C_n(t) = 0$ pentru oricare *t* şi $C_i(0) = \infty$ pentru toate $i \neq n$. Atunci

$$C_i(t+1) = \min_{k:(t,k)\in A} \{C_{ik} + C_k(t\}, i = 1,...,n-1 \qquad (7)$$

Această formulare defineşte algoritmul Bellman-Ford.

Se ataşează fiecărui nod indicatorii *π* şi *d*. De tip pointer, *π* este o locaţie în care se stochează referinţa către nodul anterior din calea cea mai scurtă, iar *d* păstrează distanţa pe calea selectată. În fiecare iteraţie pentru fiecare nod se testează toate arcele şi în cazul în care noua cale obţinută are o lungime mai mică decât cea reţinută deja atunci noua cale se reţine, corectând rezultatul. Algoritmul este deci de tipul corecţia etichetei: *π* şi *d* pot fi considerate împreună ca fiind o etichetă.

```
Algoritmul Bellman-Ford

Iniţializează: d ← ∞, d(s) ← 0, π ← nil

Pentru m ← 1 la |N|-1 Repetă
   Pentru (u,v)∈A ( adiacente din u )
      Dacă d[v]>d[u]+c(u,v) Atunci
           π ← u, d[v] ← d[u]+c(u,v)
```

Fig. 96 Algoritmul conceptual Bellman-Ford

Acest algoritm se pretează la programare dinamică.

Considerând că cea mai scurtă cale dintre origine şi nodul v are m arce şi constă dintr-o cale între origine şi un nod u care are m-1 arce, atunci:

$$\pi_m(v) = \min_{\{(u,v)\}} \pi_{m-1}(u) + c(u,v) \qquad (8)$$

Din structura sa foarte simplă se poate observa că algoritmul Bellman-Ford are complexitate $O(mn)$.

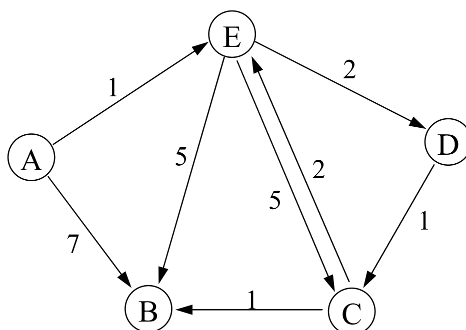


Fig. 97 Exemplu de graf pentru rezolvare cu algoritmul Bellman-Ford

	A	**B**	**C**	**D**	**E**
0	0	∞	∞	∞	∞
1	0	7(A)	∞	∞	1(A)
2	0	6(E)	6(E)	3(E)	1(A)
3	0	6(E)	4(D)	3(E)	1(A)
4	0	5(E)	4(D)	3(E)	1(A)

Fig. 98 Exemplu de rezolvare cu algoritmul Bellman-Ford

10.3. CELE MAI SCURTE K DRUMURI PE GRAF

INTRODUCERE

După cum am văzut în partea anterioară problema căii minime se reduce la a găsi acel drum între o origine şi o destinaţie care are cel mai scăzut cost. Cost are un înţeles larg putând fi: distanţă, greutate, valoare monetară, consum de combustibil, sau chiar o variabilă agregată ce înglobează diferite dezavantaje ale unei anumite căi.

În multe situaţii calea minimă singură nu descrie complet situaţia. Este necesar să se determine şi următoarele căi care unesc originea şi destinaţia (următoarele în sensul ordonării costurilor aşa cum au fost ele considerate).

Problema se poate defini deci astfel: din mulţimea căilor care leagă originea şi destinaţia să se determine primele k în ordinea crescătoare a costurilor.

Necesitatea includerii în cercetările operaţionale rezidă în faptul că, de exemplu, şoferii nu aleg obligatoriu calea cea mai scurtă din reţea. Fiind o decizie psihologică, criteriile fiind adesea subiective, alegerea se conformează legilor psihologiei şi are mai degrabă o distribuţie statistică. Combinarea cunoştinţelor despre distribuţia opţiunilor cu calculul unui număr de căi, în ordinea lor firească, poate conduce la înţelegerea comportării microscopice şi macroscopice a traficului. Rezultatul cel mai spectaculos putând fi alocarea dinamică a traficului, un subiect de mare actualitate în zonele cu trafic aglomerat.

Modificările ce apar în trafic prin dirijarea intenţionată sau prin informaţii în timp real ce parvin participanţilor conduc subsecvent la modificarea costurilor ataşate anumitor sectoare din reţea. Prin aceasta toate căile dintre două noduri oarecare din reţea vor avea condiţii diferite şi implicit ordinea în care ele sunt cotate se va modifica. Prin aceasta se modifică dinamic şi preferinţele.

Este imperios necesar să se dezvolte algoritmi de calcul a primelor *k* drumuri scurte între origine şi destinaţie. Aplicaţiile lor sunt numeroase în domeniul infrastructurii transporturilor, comunicaţii, logistică etc.

În esenţă, problema celor mai scurte k drumuri este o generalizare a problemei căii minime. Dacă pentru drumul cel mai scurt găseşte doar o cale între origine şi destinaţie, problema aceasta găseşte primele k drumuri ordonate crescător după un criteriu numit generic distanţă. Nu este aşa de simplu de rezolvat ca problema drumului minim dar are o arie mai largă de aplicaţii.

În managementul transporturilor calculul căilor cele mai scurte este o prezenţă constantă. Dacă din motive diverse (lucrări, congestii de trafic, activităţi sportive etc.) cea mai scurtă cale nu este disponibilă, este de dorit să dispunem de variante suplimentare. De asemenea, planificarea urbană trebuie să modeleze mult mai aproape de realitate fluxul vehiculelor pe variante alternative decât pe cale unică. Este cunoscut că participanţii la trafic aleg între variante cu o anumită probabilitate.

Pollack [167] a formulat problema generală după cum urmează:

Se dă un set de noduri fiecare pereche fiind conectată printr-o conexiune. Se dau distanţele dintre noduri fără să fie obligatoriu simetrice (distanţa de la nodul *i* la nodul *j* nu este obligatoriu aceeaşi ca distanţa de la *j* la *i*). Toate distanţele se consideră nenegative, dacă nu există conexiune directă distanţa este infinit. Distanţa dintre oricare pereche de noduri nu este obligatoriu o distanţă fizică (poate fi cost, timp, întârziere etc.). Calea k dintre nodurile *i* şi *j* este definită ca fiind al *k*-ulea

drumul fără bucle din lista drumurilor care unesc i şi j. Această problemă se mai numeşte şi problema ordonării căilor (paths ranking problem).

Modalitatea clasică este de a găsi toate căile dintre origine şi destinaţie, de a le sorta şi de a alege primele k. Metoda găseşte întotdeauna rezultatul corect dar este foarte laborioasă. Poate fi utilizată doar pentru reţele foarte mici. Toate celelalte metode găsesc cea mai bună soluţie doar cu o anumită ***probabilitate***. Eficienţa fiecărei metode depinde de topologia reţelei.

DIFERITE METODE DE TRATARE

Polack [167] şi Shier [172], [173] trec în revistă metodele de algoritmi dezvoltaţi pentru rezolvarea problemei primelor k drumuri. În continuare vom enumera câteva exemple.

Metoda Bock, Kantner, Haynes

Bock, Kantner, and Haynes [150] descriu o metodă de determinare a primelor k drumuri utilizând „tulpini” şi arbori. Metoda listează sistematic toate căile dintre origine şi destinaţie. După ce toate căile au fost găsite, ele se ordonează. Metoda nu poate fi folosită decât pentru reţele reduse ca dimensiune dar în acest caz este cea mai bună metodă.

Metoda drumului cel mai scurt

O metodă simplă pentru valori mici ale lui k porneşte de la aflarea celei mai scurte căi. Ideea este că orice drum scurt din listă este o deviere de la o cale mai scurtă [168]. Pentru fiecare nivel al lui k trebuie complet rezolvat nivelul anterior. De fiecare dată se consideră pe rând căile deja incluse în listă şi se elimină câte un arc după care se calculează calea minimală în graful rezultat. Se pretează doar la reţele restrânse şi valori mici ale lui k.

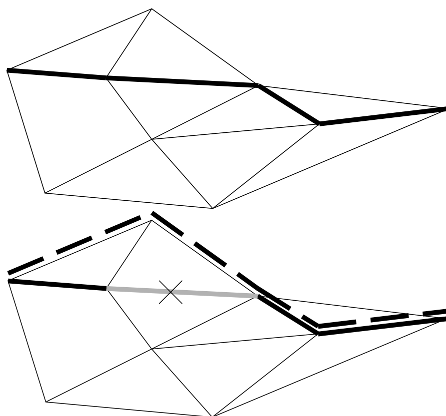


Fig. 99 Exemplificare a metodei drumului cel mai scurt

Metoda lui Bellman şi Kalaba

Bellman şi Kalaba [147] prezintă o metodă care încearcă să determine primele k drumuri de la toate nodurile către un nod dat. Algoritmul a fost evidenţiat şi perfecţionat de Pollack [167].

Prin această metodă se evaluează consecutiv calea cea mai scurtă, a doua cale, a treia şi aşa mai departe. Pentru fiecare valoare *k* nivelul anterior trebuie să fie rezolvat. În principiu, trebuie să se rezolve ecuaţiile:

$$u_i = \min_{j \neq i}\left(t_{ij} + u_i\right), \text{ unde } i=1,2,\ldots,n-1,$$

$$u_n = 0 .$$

ALGORITMI CU CORECŢIA ETICHETEI (LABEL-CORRECTING)

Petru fiecare nod se poate ataşa o „etichetă” constând într-un vector în care se păstrează fiecare din primele *j* căi şi lungimea lor. Filozofia algoritmilor cu corecţia etichetei (există o întreagă familie) constă în modificarea continuă a etichetei ataşate nodurilor până când, în final, se poate spune că valoarea ataşată este corectă [173]. Valorile de lucru ataşate sunt temporare.

Aceşti algoritmi dau primele k drumuri între un nod de origine şi toate celelalte noduri din reţea. algoritmul tipic este format din 3 paşi.

LC1. Se porneşte cu o aproximaţie iniţială a lungimii căii k de la nodul 1 la fiecare nod j. Aceasta înseamnă atribuirea unui vector $x(j)=(x_{j1}, x_{j2},\ldots,x_{jk})$ de fiecărui nod j. Există mai multe posibilităţi pentru valorile iniţiale ale vectorilor x(j) dar se recomandă ca fiind convenabile:

$x(1)=(0,\infty,\ldots,\infty)$

$x(j)=(\infty,\infty,\ldots,\infty)$ pentru j=2..n

LC2. Se alege un nou arc i şi se procesează arcul. Prin procesarea arcului (i,j) se înţelege faptul că se încearcă să se îmbunătăţească vectorul x(j) prin verificarea căilor de la până la nodul i cuprinse în vectorul i la care se adaugă arcul (i,j). Dacă oricare dintre valorile $\{x_{im}+a_{ij}:m=1\ldots k\}$ conduc la un drum mai scurt decât oricare dintre cele cuprinse în x(j), atunci vectorul x(j) este actualizat prin includerea căii mai scurte. Se subînţelege că se realizează toate actualizările posibile ale vectorului x(j) utilizând x(i) când se procesează arcul (i,j).

LC3. STOP dacă se face verificarea criteriilor de încheiere. Altfel întoarcere la pasul LC2.

Avantajele unei astfel de metode sunt: oferă soluţii exacte şi rămân valide chiar şi în cazul costurilor negative. Sigura cerinţă este să nu existe circuite negative. Mai jos sunt prezentate patru tipuri de algoritmi cu corecţia etichetei:

Forma de bază a algoritmului cu corecția etichetei

În această metodă arcele sunt prelucrate într-o ordine fixă. Arcele adiacente nodului notat cu 1 sunt prelucrate mai întâi, apoi cele adiacente nodului 2 etc.

Dacă se întâlneşte un nod cu etichetă (∞,∞,…,∞) arcele corespunzătoare nu vor fi procesate.

Dacă într-o iterație nu se face nici o actualizare atunci algoritmul se opreşte.

Pas1: Pune x(1)←(0,∞,…,∞), x(j)←(∞,∞,…,∞) pentru j≠1. Atribuie NOD=1.

Pas2: Dacă x(nod)←(∞,∞,…,∞) atunci mergi la pas3, altfel examinează NOD: pentru fiecare j adiacent față de NOD procesează arcul (NOD,j).

Pas3: Dacă NOD<N atunci NOD←NOD+1 şi treci la Pas2.

Pas4: Dacă s-au făcut modificări asupra unei etichete x(j) atunci NOD←1 şi treci la Pas2. Altfel STOP.

Algoritm cu indicator de modificare (alteration flag – AF)

Este o variantă a algoritmului de bază. Ideea este ca într-o iterație să nu se examineze un nod dacă vectorul celor k-drumuri asociat nu s-a modificat de la examinarea anterioară. Acest algoritm asociază pentru fiecare nod un indicator $T(j)$ cu proprietatea că $T(j)=0$ dacă nu au avut loc schimbări, $T(j)=1$ a apărut o schimbare.

Pas1: x(1)=(0,∞,…,∞), x(j)=(∞,∞,…,∞) pentru j≠1; T(1)=1, T(j)=0 pentru j≠1. Stabileşte NOD=1

Pas2: Dacă T(NOD)=0 atunci treci la PAS3. Altfel, începe examinarea lui NOD: pentru toate nodurile j adiacente lui NOD, procesează arcul (NOD,j). Dacă există o modificare la eticheta nodului j atunci T(j)=1. T(NOD)=0.

Pas3: Dacă NOD<N atunci NOD←NOD+1 şi trece la PAS2. Altfel, verifică dacă în PAS2 a fost modificat un vector x(j). Dacă au fost făcute modificări NOD=1 trece la PAS2. Altfel STOP

Algoritmul cu listă de secvențe (sequence list – SL)

Nodurile nu mai sunt examinate într-o ordine fixă. Se generează o listă de noduri şi de fiecare dată se examinează următorul din listă.

Când este examinat un nod, toate arcele adiacente din acel nod sunt procesate şi acel nod este eliminat din listă. Lista este de tipul FIFO. Când un arc *(i,j)* este procesat şi rezultă o modificare a etichetei nodului j acesta este adăugat în listă, dacă nu este deja.

În principiu algoritmul are următoarea formă:

PAS1: x(1)=(0,∞,…,∞), x(j)=(∞,∞,…,∞) pentru j≠1, adaugă nodul 1 în listă.

PAS2: Extrage nodul i din capul listei. Examinează nodul i: pentru toate nodurile j adiacente dinspre i procesează nodul (i,j). Dacă nu este deja în listă adaugă j în coada listei.

PAS3: Dacă mai există noduri în listă trece la PAS2. altfel STOP.

Algoritmul cu dublă baleiere (double-sweep – DS)

Metoda reprezintă o altă versiune a algoritmului de bază. În loc să se examineze nodurile în aceeaşi ordine 1,2,...,n în fiecare iteraţie, această metodă face două treceri alternante înainte şi înapoi. Într-o primă fază se consideră nodurile *j* în ordinea 1,2,...,n şi se examinează arcele (j,i) pentru i<j. În faza a doua se consideră nodurile *j* în ordinea n,...,2,1 şi se examinează nodurile (j,i) pentru j>i. Procesul continuă până când nu mai există alte iteraţii posibile.

PAS1: x(1)=(0,∞,…,∞), x(j)=(∞,∞,…,∞) pentru j≠1.

PAS2: Consideră NOD=2,…,N. Pentru nodurile i, i<NOD, care sunt adiacente cu NOD, procesează arcul (NOD,i)

PAS3: Dacă nici o eticheta x(j) nu s-a modificat în timpul PAS2 atunci STOP.

PAS4: Consideră NOD=N-1,…,1. Pentru nodurile i, i>NOD, care sunt adiacente cu NOD, procesează arcul (NOD,i).

PAS5: Dacă nici o eticheta x(j) nu s-a modificat în timpul PAS4 atunci STOP. Altfel trece la PAS2.

ALGORITMUL CU FIXAREA ETICHETEI (LABEL-SETTING – LS)

Acest tip de algoritm este adecvat calculării primelor k drumuri pentru reţele cu arce cu lungimi ne-negative. Prin contrast cu algoritmii cu corecţia etichetei, algoritmul cu fixarea etichetei identifică în fiecare pas o valoare corectă şi permanentă pentru un nod.

PAS1: x(1)=(0,∞,…,∞), x(j)=(∞,∞,…,∞) pentru j≠1. adaugă toate nodurile în listă (toate componentele sunt făcute temporare) şi atribuie i=1

PAS2: Găseşte cel mai mic component temporar TEMP pentru nodul i. Pentru fiecare nod j adiacent dinspre i adaugă, dacă este posibil, valoarea TEMP+A(i,j), în vectorul x(j).

PAS3: Defineşte ca permanentă componenta TEMP a nodului i. Înlătură nodul i din listă dacă nu mai sunt componente temporare pentru acest nod.

PAS4: Dacă lista este goală atunci STOP. Altfel, găseşte un nod i a cărui cea mai mică componentă este minimă pentru toate nodurile din listă şi trece la PAS2.

10.4. BIBLIOGRAFIE

[144] Ahuja R.K., Magnanti T.L., Orlin J.B.: *Network flows: Theory, Algorithms, and Applications*; Prentice Hall, Englewood Cliffs, NJ, 1993.
[145] Ahuja R.K., Mehlhorn K., J. B. Orlin, R. E. Tarjan (1990): *Faster algorithms for the shortest path problem*; Journal of ACM 37, 213-223.
[146] Ahuja R.K., Orlin J.B., S. Pallottino, M. G. Scutellà: *Minimum time and minimum cost path problems in street networks with periodic traffic lights*. Transportation Science 2000.
[147] Bellman, R., Kalaba R.: *On kth best policies*; J. Soc. Indust. Appl. Math. 8 no. 8, pp.582-588 1960.
[148] Bertsekas D.P. (a): *An Auction algorithm for shortest paths*; SIAM Journal on Optimization 1, pp.425-447, 1991.
[149] Bertsekas D.P. (b), *Linear network optimization: Algorithms and codes*. M.I.T. Press, Cambridge, MA, 1991.
[150] Bock F., Kantner H., Haynes J.: *An algorithm (The r-th best path algorithm) for finding and ranking paths through a network.*; Research Report, Amour research Foundation, Chicago, IL, November 15, 1957.
[151] Cherkassky B. V., A. V. Goldberg, T. Radzik: *Shortest paths algorithms: Theory and experimental evaluation*; Research project, Department of Computer Science, Cornell and Stanford Universities and Krasikova Institute for Economics and Mathematics. 1993, Mathematical Programming 73, pp.129-174, 1996.
[152] Denardo E. V., B. L. Fox: *Shortest-route methods: 1. Reaching, pruning, and buckets*; Operations Research 27 (1), pp.161-186, 1979.
[153] Denardo E. V., B. L. Fox; *Shortest-route methods: 2. Group knapsacks, expanded networks, and branch-and-bound*; Operations Research 27 (3), pp.548-566, 1979.
[154] Deo N., C. Pang: *Shortest path algorithms: Taxonomy and annotation*; Networks 14 , pp.275-323, 1984.
[155] Diestel Reinhelt: Graph Theory; Electronic Edition 2000, Springer-Verlag, New York 2000.
[156] Ertl G.(1996) Optimierung und Kontrolle, Shortest Path Calculations in Large Road Networks. Project in Discrete Optimisation, Karl-Franzens-Universitat Graz.
[157] Gallo G. (1980), "Reoptimization procedures in shortest path problems", Rivista di Matematica per le Scienze Economiche e Sociali 3, 3-13.
[158] Gallo G., S. Pallottino: *A new algorithm to find the shortest paths between all pairs of nodes*; Discrete Applied Mathematics 4, pp.23-35, 1982.
[159] Gallo G., S. Pallottino: *Shortest path methods in transportation models*; in (M. Florian, ed.) Transportation planning models, North-Holland, pp.227-256, 1984.
[160] Gallo G., S. Pallottino; *Shortest path algorithms*; Annals of Operations Research 13, pp.3-79, 1988.
[161] Gallo G., S. Pallottino; *Shortest path methods: A unifying approach*; Mathematical Programming Study 26, pp.38-64, 1986.
[162] Hart P E, Nilson N J.: *A formal basis of the heuristic determination of minimum cost paths*; IEEE Transactions of Systems Science and Cybernetics 4(2), 100-107, 1968.

[163] Johnson D. B. (a): *Algorithms for shortest paths*, Ph.D. Thesis, Cornell Univ., tr-73-169. 1973.
[164] Johnson D. B. (b), *A note on Dijkstra's shortest path algorithm*; Journal of the ACM 20(3), pp.385-388, 1973.
[165] Pallottino Stefano, Scutellà Maria Grazia: *Shortest Path Algorithms in Transportation models: classicals and inovative aspects*; Technical Report TR-97-06, Università di Pisa, Dipartamento di Informatica, Pisa Italia, 1997.
[166] Pearsons J.: *Heuristic Search in Route Finding*; Master's Thesis, University of Auckland, 1998.
[167] Pollack Maurice: *Solutions of the kth bets through a network – Review*; Journal of mathematical analysis and applications 3, pp.547-559, 1961.
[168] Pollack Maurice: *The k-th best route trough a network*; Operations Research,9, no.4, 578-580 (1961);
[169] Scînteie Rodian: *Development of an algorithm for k-shortest path*; Simpozion cu participare internațională, Iaşi, Octombrie 2003.
[170] Sedgewick R and Vitter J S.: *Shortest Paths in Euclidean Graphs*. Algorithmica 1, pp.31-48, 1986.
[171] Shier D. R., C. Witzgall: *Properties of labeling methods for determining shortest path trees*; Journal of Research of the National Bureau of Standards 86, pp.317-330, 1981.
[172] Shier D.R.: *Iterative Methods for determining the k shortest paths in a network*; Networks, 6: pp.205-229, 1976.
[173] Shier D.R.: *On algorithms for finding the k shortest paths in a network*; Networks, 9: pp.195-214, 1979.
[174] Steenbrink P. A.: *Optimization of transport networks*, J. Wiley, London 1974.
[175] Tarjan R. E.: *Data structures and network algorithms*; SIAM, Philadelphia PA 1983.
[176] Van Vliet D.: *Improved shortest path algorithms for transport networks*; Transportation Research 12, pp.7-20, 1978.

ISBN 973-86343-3-4
Editura Societăţii Academice „Matei-Teiu Botez”
Iaşi 2003

Capitolul 11

ELEMENTE DE TEORIA COZILOR

11.1. INTRODUCERE

Cozile de aşteptare, numite şi „*fire de aşteptare*” de unii autori de limbă română (Turbuţ-1988 [215]), sunt un aspect uzual în viaţa cotidiană. Într-o formă sau alta fiecare dintre noi suntem afectaţi de existenţa lor. Fie că aşteptăm la rând la magazin fie că suntem în maşină la un semafor, ne supunem aceloraşi legi ale cozilor.

Scopul studierii cozilor de aşteptare este de a stabili structura sistemului, a determina parametrii funcţionali (timpul mediu de aşteptare în sistem, media lungimii cozii, utilizarea punctelor de deservire) şi de a identifica măsurile adecvate de optimizare.

Teoria matematică utilizată în studiul cozilor este o aplicaţie a teoriei probabilităţilor şi se întâlneşte sub diferite denumiri: teoria traficului, teoria cozilor, teoria congestiei, teoria servirii de masă, teoria sistemelor stocastice de deservire etc. (Cooper-1981 [187]). Termenul de teoria traficului se aplică atât traficului de informaţii în telefonie şi telecomunicaţii cât şi traficului de vehicule.

Uneori studiul teoriei cozilor se aplică în sisteme în care formarea acestora nu este permisă, ceea ce pare impropriu, dar tocmai acest tip de analiză specifică poate conduce la luarea deciziilor adecvate care să menţină sistemul funcţional.

Structura generală a unei cozi de aşteptare este reprezentată în figura următoare:

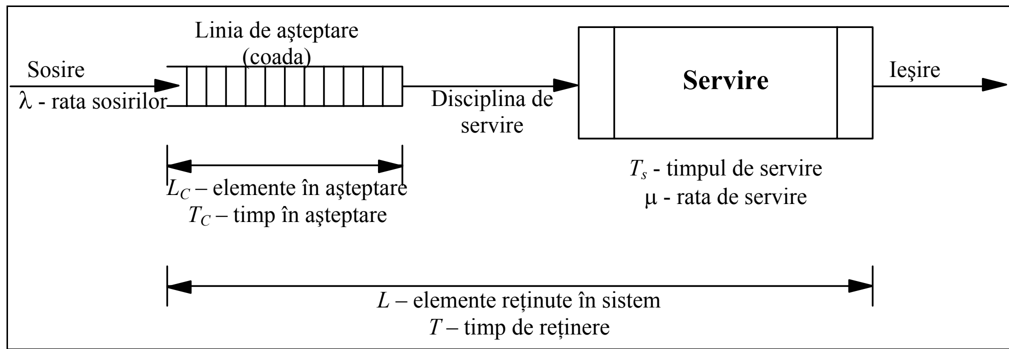


Fig. 100 Structura unei sistem cu coadă de aşteptare

Istoric vorbind, subiectul a fost analizat şi dezvoltat din punct de vedere matematic în contextul ingineriei traficului telefonic. În prezent este aplicat pe larg în reţele de calculatoare, cercetări operaţionale, inginerie civilă etc. De remarcat că tehnicile matematice de analiză sunt similare cu cele utilizate în domenii ce nu par a fi înrudite precum inventarul stocurilor de producţie, studiul amenajărilor hidrotehnice, asigurări etc.

NOTAŢIA KENDALL

Aşa-numita notaţie Kendall este o schemă de caracterizare a cozii de aşteptare. Prin aceasta se descrie atât componenţa sistemului cât şi comportarea acestuia.

Notaţia Kendall presupune existenţa a şase parametri:

1. distribuţia sosirilor;
2. distribuţia servirii;
3. numărul de puncte de deservire;
4. capacitatea maximă;
5. populaţia totală;
6. disciplina servirii.

Aceşti parametri sunt cuprinşi în formula:

$$A/B/m/N/P-S \tag{9}$$

Primii trei parametri au fost trataţi de Kendall în 1953 [198], următorii au fost introduşi ulterior.

Distribuţia sosirilor - A reprezintă legea statistică a intervalului de timp după care „clienţii" sosesc la coadă

Distribuţia servirii B reprezintă legea statistică a timpului de tratare a fiecăruia în cadrul punctului de deservire sau procesare.

m este numărul de puncte de deservire sau procesare care sunt caracterizate de distribuția B.

N numărul total de locuri în coadă inclusiv în punctele de deservire. Dacă nu se specifică altfel se consideră infinit.

P este populația totală sau maximul de entități care poate să se aşeze la coadă (ex. nu putem să ne aşteptăm să găsim la un semafor mai multe maşini decât s-au produs). Dacă nu se specifică explicit altfel acest parametru se consideră infinit.

S - disciplina de deservire reprezintă ordinea în care entitățile din coadă sunt selectate din linia de aşteptare şi tratate. Acestea pot fi: FCFS – primul venit primul servit; LCFS – ultimul venit primul tratat; SPT – prioritate de servire funcție de timp, unde selecția se face pe baza timpului de tratare estimat (de exemplu la băcănie când cei care cumpără doar un singur articol sunt serviți peste rând indiferent de momentul sosirii); SIRO servire în ordine aleatoare; prioritate statică – fiecărui tip de solicitare i se asociază o prioritate care nu se modifică în timp. În cadrul diferitelor nivele de prioritate disciplina poate fi FCFS, LCFS, SPT etc.; prioritate dinamică – prioritatea asociată poate să se modifice.

Se pot utiliza şi alte tipuri de priorități considerate convenabil funcție de scopul urmărit. Dacă nu se indică altfel se implică o disciplină de tipul FCFS / FIFO.

Pentru parametrii A şi B este comună utilizarea următoarelor notații:

- M (Markov) – presupune o distribuție a intervalelor de sosire de tip exponențial de forma $A(t) = 1 - e^{-\lambda t}$ sau $a(t) = \lambda e^{-\lambda t}$ unde $\lambda > 0$ este parametrul de formă. Avantajul unei astfel de distribuții în descrierea comportamentului cozii constă în faptul că este singura distribuție continuă care este fără memorie (proprietatea Markov) adică intervalul şi probabilitatea sosirilor nu depinde de timpul scurs şi de istoricul sosirilor anterioare (proces Poisson).
- D (Deterministic) – toate valorile sunt constante. Se presupune că sosirile au loc la intervale exacte de timp sau că servirea durează un timp fix.
- E_r (Erlang-*r*) – Distribuția Erlang cu *r* faze. Constanta *r* este mai mare ca zero. Forma generală a distribuției Erlang-*r* este:

$$F(t) = 1 - e^{-r\mu t} \sum_{i=0}^{r-1} \frac{(r\mu t)^i}{i!}, \ \mu > 0 \tag{10}$$

- H_k (Hiper-*k*): Distribuția hiper-exponențială cu *k* faze. Forma generală este

$$F(t) = \sum_{i=1}^{k} q_i \left(1 - e^{-\mu_i t}\right); \qquad \mu_i > 0 \, ; q_i > 0 \, ; \sum_{i=1}^{k} q_i = 1 \tag{11}$$

- *G* (General) – distribuţia nu este specificată. Poate avea oricare formă. Se cunoaşte doar media şi varianţa.
- *GI* (General cu timp independent între sosiri) – distribuţia nu este specificată. Poate avea oricare formă. Se cunoaşte doar media şi varianţa.

PROCESUL POISSON

Procesul Poisson este unul dintre cele mai importante modele comportamentale utilizate în teoria cozilor de aşteptare.

Un proces Poisson este un proces care satisface următoarele condiţii:

- Numărul de modificări ale stării în intervale nesuprapuse sunt independente pentru toate intervalele.
- Probabilitatea să existe exact o singură schimbare de stare într-un interval suficient de mic $\tau = \frac{1}{n}$ este $\pi_\tau = \nu\tau$, unde ν este probabilitatea unei schimbări şi n este numărul de încercări.
- Probabilitatea ca într-un interval suficient de mic τ să aibă loc două sau mai multe schimbări este 0.

Dacă se consideră un timp suficient de îndelungat, pe intervale discrete, distribuţia care îndeplineşte aceste condiţii se numeşte distribuţia Poisson cu probabilitatea $P(N(t) = n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$. Considerând distribuţiile pe un domeniu continuu, singura care îndeplineşte condiţiile de mai sus este distribuţia exponenţială cu funcţia distribuţie a probabilităţii timpului dintre două sosiri $f(t) = P(N(t) = 1) = e^{-\lambda t}$. În plus acestea sunt şi distribuţii fără memorie (proprietatea Markov).

Proprietăţile unui proces Poisson sunt:

- Superpoziţia;
- Selecţia aleatoare;
- Despărţirea aleatoare.

Aceste proprietăţi ale proceselor Poisson sunt importante în raţionamentele utilizate în demonstraţiile legate de proprietăţile cozilor de aşteptare.

Superpoziţia – Dacă se consideră două procese Poisson complet independente de parametru λ_1 respectiv λ_2 atunci suprapunerea celor două procese va fi tot un proces Poisson cu parametru $\lambda = \lambda_1 + \lambda_2$.

Superpoziţia este prezentată în principiu în figura următoare:

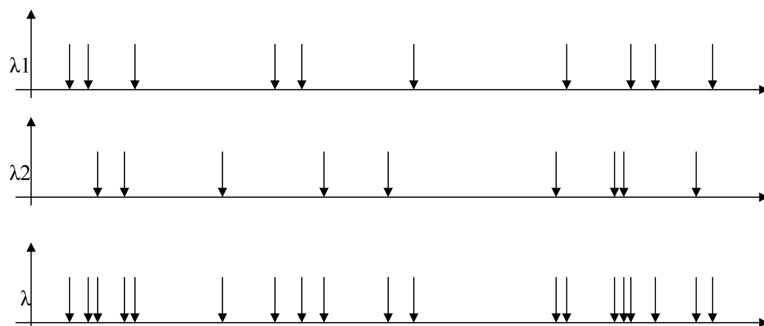


Fig. 101 Superpoziția a două procese Poisson

Selecția aleatoare – dacă dintr-un proces Poisson de parametru λ se face o selecție aleatoare în aşa fel încât pentru fiecare sosire probabilitatea de selecție este π, independentă de celelalte, atunci procesul rezultat este tot un proces Poisson de parametru $\pi\lambda$.

Principiul selecției aleatoare este prezentat în figura următoare:

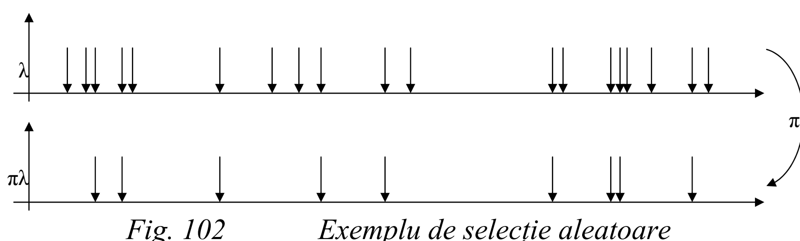


Fig. 102 Exemplu de selecție aleatoare

Despărțirea aleatoare (random split) – dacă se consideră un proces Poisson de parametru λ şi se desparte în două subprocese cu probabilitățile π_1 şi π_2 astfel încât $\pi_1 + \pi_2 = 1$ cele două procese rezultate sunt ***procese Poisson independente*** de parametru $\pi_1\lambda$ şi $\pi_2\lambda$.

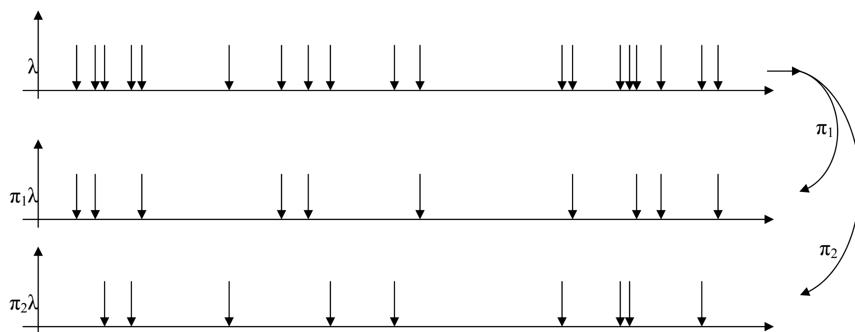


Fig. 103 Despărțire aleatoare

PROPRIETĂŢI ALE COZILOR DE AŞTEPTARE

Rata de ocupare

O caracteristică uzuală a cozii este utilizarea sau rata de ocupare.

Rata de ocupare reprezintă proporţia de timp cât există cereri sau clienţi în sistemul cu coadă de aşteptare.

Pentru sisteme cu un singur operator de servire în care distribuţia sosirilor şi a servirii sunt indiferente, dar pentru care rata sosirilor este λ şi media timpului de servire este $E[B]$, proporţia de timp în care se procesează cererea (se efectuează servirea) este dată de $\lambda E[B]$. Deoarece proporţia nu poate să depăşească valoarea 1 cerinţa de stabilitate este $\lambda E[B] < 1$.

În general această valoare calculată se numeşte utilizare sau rată de ocupare şi se notează cu ρ. Pentru sistemele cu mai multe unităţi de servire (*G/G/c*) rata de ocupare este:

$$\rho = \frac{\lambda E[B]}{c} < 1 \tag{12}$$

Legea lui Little

Această proprietate a fost intuită anterior dar a fost sistematizată şi demonstrată de Little în 1961 [202].

Considerăm un sistem în echilibru în care sosesc clienţi, aşteaptă un interval de timp în sistem şi pleacă. Dacă notăm rata sosirilor cu λ, timpul mediu de aşteptare în sistem cu T şi numărul mediu de clienţi cu L atunci relaţia dintre cele trei este:

$$L = \lambda T \tag{13}$$

Demonstraţia lui Little, era neintuitivă, destul de complicată şi necesita un aparat matematic dificil. Ulterior Stidham 1974 a reuşit să facă o demonstraţie simplă şi riguroasă care impune doar ca λ şi T să existe şi să fie finite.

Să considerăm:

$L(t)$ numărul de clienţi din sistem la momentul *t*;

$a(t)$ numărul de clienţi care au sosit în intervalul $[0, t]$;

$b(t)$ numărul de clienţi care au plecat în intervalul $[0, t]$;

T_i timpul pe care clientul cu numărul *i* îl petrece în sistem.

Media $L(\tau)$ până la momentul *t* este:

$$L_t = \frac{1}{t}\int_0^t L(\tau)d\tau \tag{14}$$

Rata medie a sosirilor din intervalul $[0, t]$ este:

$$\lambda_t = \frac{a(t)}{t} \tag{15}$$

Timpul mediu pe care un client îl petrece în sistem este:

$$T_t = \frac{1}{a(t)}\sum_{i=0}^{a(t)} T_i \tag{16}$$

Reluând prima ecuație şi ținând seama de egalitatea $\int_0^t L(\tau)d\tau = \sum_{i=0}^{a(t)} T_i$ obținem:

$$L_t = \frac{1}{t}\int_0^t L(\tau)d\tau = \frac{a(t)}{t}\frac{\int_0^t L(\tau)d\tau}{a(t)} = \lambda_t T_t \tag{17}$$

Dacă acest calcul se face după un timp suficient de lung ($t \to \infty$) atunci $L_t \to L$, $\lambda_t \to \lambda$ şi $T_t \to T$.

Deci pe termen lung:

$$L = \lambda T \tag{18}$$

Ceea ce era de demonstrat.

Teorema lui Little este indispensabilă în analiza sistemelor cu coadă de aşteptare.

Relația este valabilă şi între timpul mediu de aşteptare pentru a fi servit T_C şi numărul mediu de elemente care aşteaptă să fie servite L_C.

Proprietatea PASTA

Cozile de aşteptare cu sosire de tip Poisson $(M/\bullet/\bullet)$ posedă o proprietate specială: proporția celor care sosesc şi găsesc sistemul într-o stare *S* este exact aceeaşi cu proporția timpului cât sistemul este în starea *S* considerată. Proprietatea nu este

adevărată în general. Această proprietate se numeşte PASTA - Poisson Arrivals See Time Averages (sosirile Poisson sesizează timpul mediu) ([223], [224]).

Proprietatea PASTA este un instrument puternic în analiza şi calculul caracteristicilor sistemelor cu coadă de aşteptare.

11.2. ANALIZA COZILOR DE AŞTEPTARE

Cozile pot fi stabile sau instabile. Sunt stabile acele sisteme în care procesul evoluează într-o manieră predictibilă. Dacă solicitarea este mai mică decât capacitatea de servire sistemul poate atinge echilibrul şi putem găsi prin calcul modele comportamentale. Dacă solicitarea este mai mare decât capacitatea atunci apar congestii, blocaje şi sistemul se comportă haotic.

Analiza cozii de aşteptare presupune evaluarea caracteristicilor de sosire şi de ieşire şi stabilirea comportamentului sistemului în fiecare dintre stări sau cel puţin calcularea timpului mediu de aşteptare în sistem, numărului mediu de elemente din sistem, timpului mediu de aşteptare pentru a fi servit şi numărul mediu de elemente care aşteaptă să fie servite.

Avem deci patru variabile L, T, L_C, T_C şi trei ecuaţii pe care le putem utiliza:

$$L = \lambda T \,,\ L_C = \lambda T_C \,,\ T = T_C + E[B] = T_C + \frac{1}{\mu} \qquad (19)$$

Unde $E[B]$ este timpul mediu de servire şi μ este rata de servire.

Pentru a afla cele patru valori mai este necesară o ecuaţie. Cel mai adesea se utilizează:

$$L = \sum_{i=0}^{\infty} (i\pi_i) \qquad (20)$$

Unde π_i este probabilitatea ca în sistem să existe exact i clienţi.

MODELE DE COZI DE AŞTEPTARE

O coadă de aşteptare poate fi privită ca un proces de generare şi dispariţie (birth-and-death process). Un model adecvat pentru o astfel de situaţie este lanţul Markov.

Acest model constă în reprezentarea stărilor pe care le poate lua sistemul şi ataşarea probabilităţilor de tranziţie între stări. O reprezentare generală printr-o diagramă a tranziţiilor este schiţată în figura următoare.

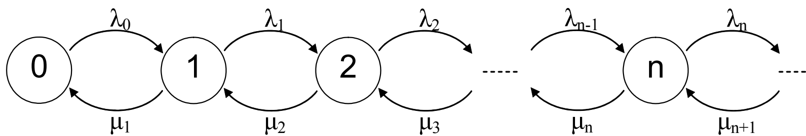


Fig. 104 Diagrama stărilor pentru un sistem cu coadă de aşteptare

În diagramă numărul stării reprezintă numărul total de elemente aflate în sistemul cu coadă de aşteptare. Verificarea stabilității presupune existența unor ecuații de echilibru care să nu fie dependente de timp. Echilibrul este dinamic în sensul că $\pi_n(t)$ - probabilitatea ca sistemul să se afle în starea n este constantă.

După cum s-a considerat în diagramă, pentru starea n sistemul are doar trei posibilități de tranziție în intervalul Δt:

1. să treacă în starea $n+1$ cu o probabilitate $\lambda_n \Delta t$;
2. să treacă în starea $n-1$ cu o probabilitate $\mu_n \Delta t$;
3. să rămână în starea n cu o probabilitate $1-(\lambda_n+\mu_n)\Delta t$.

Ecuația cu diferențe care determină probabilitatea devine:

$$\pi_n(t+\Delta t)=\pi_{n+1}(t)\mu_{n+1}\Delta t+\pi_{n-1}(t)\lambda_{n-1}\Delta t+\pi_n(t)[1-(\lambda_n+\mu_n)\Delta t] \quad (21)$$

Schimbând ce trebuie schimbat se obține:

$$\frac{d\pi_n(t)}{dt}=\pi_{n+1}(t)\mu_{n+1}+\pi_{n-1}(t)\lambda_{n-1}-\pi_n(t)(\lambda_n+\mu_n) \quad (22)$$

Dacă $n=0$ se aplică formula:

$$\frac{d\pi_0(t)}{dt}=\mu_1\pi_1(t)-\lambda_0\pi_0(t) \quad (23)$$

Sistemul de ecuații de mai sus se numeşte „ecuațiile Chapman-Kolmogorov”.

DISTRIBUȚIA PROBABILITĂȚILOR ÎN STARE STABILĂ

Pentru ca sistemul să fie în echilibru (stare stabilă) este necesar ca $\pi_n(t)=\pi_n$ constant, adică $\frac{d\pi_n(t)}{dt}=0$. Ecuațiile Chapman-Kolmogorov devin:

$$\pi_n(\lambda_n+\mu_n)=\pi_{n-1}\lambda_{n-1}+\pi_{n+1}\mu_{n+1} \quad (24)$$

pentru $n=1,2,3,4,...$ şi

$$\lambda_0 \pi_0 = \mu_1 \pi_1 \tag{25}$$

pentru $n = 0$.

Pornind de la ecuaţia (25) şi rearanjând convenabil ecuaţiile (24) $n = 1,2,3,4,...$ se obţine:

$$\pi_n \lambda_n = \pi_{n+1} \mu_{n+1} \tag{26}$$

Din ecuaţiile (25) şi (26) se pot calcula probabilităţile π_n. Astfel:

$$\pi_1 = \frac{\lambda_0}{\mu_1}\pi_0,\ \pi_2 = \frac{\lambda_0\lambda_1}{\mu_1\mu_2}\pi_0,\ \pi_3 = \frac{\lambda_0\lambda_1\lambda_2}{\mu_1\mu_2\mu_3}\pi_0,\ ...,\ \pi_n = \frac{\lambda_0\lambda_1\cdots\lambda_{n-1}}{\mu_1\mu_2\cdots\mu_n}\pi_0 \tag{27}$$

Evident

$$\sum_{n=0}^{\infty} \pi_n = 1 \tag{28}$$

Notând $\frac{\lambda_0\lambda_1\cdots\lambda_{n-1}}{\mu_1\mu_2\cdots\mu_n}$ cu Q_n rezultă:

$$\pi_0\left(1 + \sum_{n=1}^{\infty} Q_n\right) = 1 \tag{29}$$

Deci probabilitatea ca în sistem să nu fie nici un client este dată de:

$$\pi_0 = 1 \Big/ \left(1 + \sum_{n=1}^{\infty} Q_n\right) \tag{30}$$

Verificarea stabilităţii este dată de $\sum_{n=1}^{\infty} Q_n < \infty$.

11.3. COADA DE AŞTEPTARE *(M/M/1)*

Sistemul de $(M/M/1)$ este caracterizat de o rată egală a sosirii şi a servirii indiferent de starea în care se află.

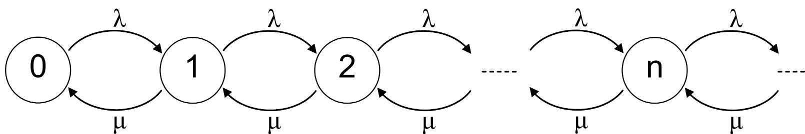


Fig. 105 Diagrama tranziţiilor pentru (M/M/1)

Aplicând ecuațiile (24) şi (25) rezultă:

$$\lambda\pi_0 = \mu\pi_1 \tag{31}$$

şi

$$\pi_n(\lambda+\mu) = \pi_{n-1}\lambda + \pi_{n+1}\mu \tag{32}$$

Înlocuind corespunzător obținem:

$$\pi_1 = \frac{\lambda}{\mu}\pi_0\,,\ \pi_2 = \frac{\lambda^2}{\mu^2}\pi_0,\ \pi_3 = \frac{\lambda^3}{\mu^3}\pi_0\,, \dots,\ \pi_n = \frac{\lambda^n}{\mu^n}\pi_0\,, \dots \tag{33}$$

Din ecuația (29) şi ținând cont de faptul că $\frac{\lambda}{\mu} = \rho$ (ocuparea) rezultă:

$$1 = \pi_0\left(1 + \sum_{n=1}^{\infty}\frac{\lambda^n}{\mu^n}\right) = \pi_0\left(\sum_{n=0}^{\infty}\rho^n\right) = \pi_0 \lim_{n\to\infty}\frac{1-\rho^n}{1-\rho} \tag{34}$$

Evident sistemul este stabil doar dacă $\rho < 1$ caz în care

$$1 = \pi_0\frac{1}{1-\rho}\ , \tag{35}$$

şi deci

$$\pi_0 = 1-\rho\ ,\ \pi_n = \rho^n(1-\rho) \tag{36}$$

Pentru calculul numărului mediu de elemente aflate în sistem (aflate în aşteptare şi în servire) vom face suma numărului de elemente din fiecare stare ponderate cu probabilitatea stării:

$$\begin{aligned} L &= \sum_{n=0}^{\infty} n\pi_n = \sum_{n=0}^{\infty} n\rho^n(1-\rho) = (1-\rho)\rho\sum_{n=0}^{\infty} n\rho^{n-1} = (1-\rho)\rho\frac{d}{d\rho}\sum_{n=0}^{\infty}\rho^n \\ &= (1-\rho)\rho\frac{d}{d\rho}\left(\frac{1}{1-\rho}\right) = (1-\rho)\rho\frac{1}{(1-\rho)^2} \end{aligned} \tag{37}$$

ceea ce conduce la formula:

$$L = \frac{\lambda}{(\mu-\lambda)} \tag{38}$$

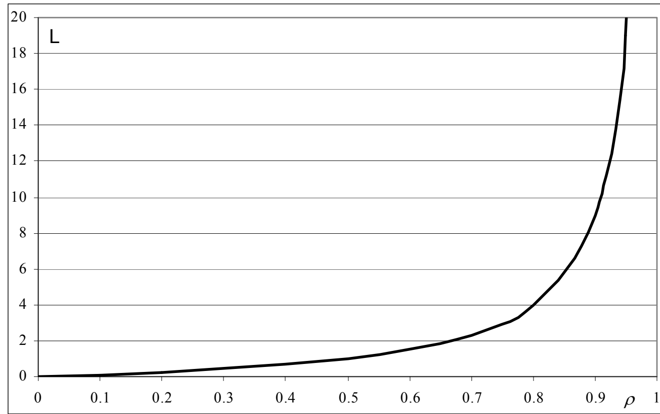


Fig. 106 *Distribuţia numărului mediu de cereri în sistem funcţie de rata de ocupare*

Aplicând formula lui Little timpul mediu de aşteptare în sistem este:

$$T = \frac{L}{\lambda} = \frac{\lambda}{(\mu - \lambda)\lambda} = \frac{1}{(\mu - \lambda)} \tag{39}$$

Timpul mediu petrecut în aşteptarea servirii este timpul mediu de aşteptare în sistem din care se scade timpul mediu de servire:

$$T_C = T - \frac{1}{\mu} = \frac{1}{(\mu - \lambda)} - \frac{1}{\mu} = \frac{\lambda}{(\mu - \lambda)\mu} \tag{40}$$

De unde lungimea medie a cozii în aşteptarea servirii este:

$$L_C = \lambda T_C = \lambda \frac{\lambda}{(\mu - \lambda)\mu} = \frac{\lambda^2}{(\mu - \lambda)\mu} \tag{41}$$

11.4. COADA DE AŞTEPTARE *(M/M/c)*

Sistemul de $(M/M/c)$ este caracterizat prin prezenţa unui număr de posturi de servire identice. Rata sosirii este constantă indiferent de starea în care se află sistemul dar rata de servire variază. Principiul este prezentat în diagrama tranziţiilor din figura următoare.

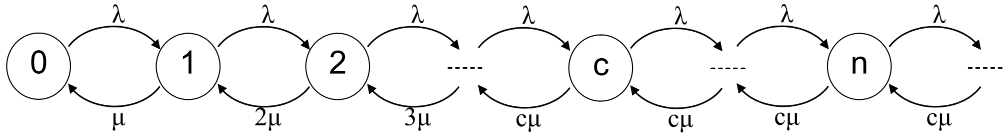


Fig. 107 Diagrama tranzițiilor pentru (M/M/c)

Rata servirii este diferită deoarece pe măsură ce există mai multe cereri există şi posibilitatea de a le trata paralel până când toate posturile de servire sunt ocupate.

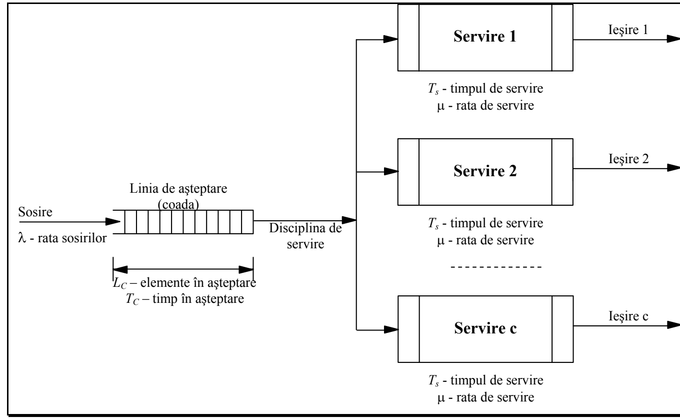


Fig. 108 Structura unei sistem cu coadă de aşteptare (M/M/c)

Conform (25) şi (26) relațiile între probabilități sunt:

$$\pi_0\lambda = \pi_1\mu \,, \lambda\pi_1 = 2\mu\pi_2 \,, \lambda\pi_2 = 3\mu\pi_3 \,,..., \lambda\pi_{c-1} = c\mu\pi_c \,,$$
$$\lambda\pi_c = c\mu\pi_{c+1} \,,..., \lambda\pi_{n-1} = c\mu\pi_n \tag{42}$$

Înlocuind iterativ corespunzător obținem:

$$\pi_1 = \pi_0 \frac{\lambda}{\mu} \,, \pi_2 = \pi_1 \frac{\lambda}{2\mu} = \pi_0 \frac{\lambda}{\mu}\frac{\lambda}{2\mu} = \pi_0 \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2 ,$$
$$\pi_3 = \pi_2 \frac{\lambda}{3\mu} = \pi_0 \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2 \frac{\lambda}{3\mu} = \pi_0 \frac{1}{3!}\left(\frac{\lambda}{\mu}\right)^3 ,..., \pi_c = \pi_0 \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c , \tag{43}$$
$$\pi_{c+1} = \frac{\lambda}{c\mu}\pi_c = \pi_0 \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \frac{\lambda}{c\mu} = \pi_0 \frac{1}{c!c}\left(\frac{\lambda}{\mu}\right)^{c+1} ,..., \pi_n = \pi_0 \frac{1}{c!c^{n-c}}\left(\frac{\lambda}{\mu}\right)^n ,...$$

sau altfel spus:

$$\pi_n = \pi_0 \frac{1}{n!}(\lambda/\mu)^n \text{ pentru } n = 0,...,c \tag{44}$$

şi

$$\pi_n = \pi_0 \frac{1}{c!c^{n-c}}(\lambda/\mu)^n \text{ pentru } n > c \tag{45}$$

Ecuația (28) devine:

$$\begin{aligned} 1 = \sum_{n=0}^{\infty}\pi_n &= \pi_0 \sum_{n=0}^{c-1}\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{\pi_0}{c!}\left(\frac{\lambda}{\mu}\right)^c \sum_{n=c}^{\infty}\left(\frac{\lambda}{c\mu}\right)^{n-c} \\ &= \pi_0\left[\sum_{n=0}^{c-1}\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \frac{c\mu}{c\mu-\lambda}\right] \end{aligned} \tag{46}$$

Probabilitatea ca în sistem să nu fie nici un client este:

$$\pi_0 = \left[\sum_{n=0}^{c-1}\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{1}{c!}\left(\frac{\lambda}{\mu}\right)^c \frac{c\mu}{c\mu-\lambda}\right]^{-1} \tag{47}$$

Prezintă importanță în cazul $(M/M/c)$ situația când toate posturile de servire sunt ocupate şi un nou sosit trebuie să aştepte.

$$\pi_w = \sum_{n=c}^{\infty}\pi_n = \frac{\pi_0}{c!}\left(\frac{\lambda}{\mu}\right)^c \sum_{n=c}^{\infty}\left(\frac{\lambda}{c\mu}\right)^{n-c} = \frac{\pi_0}{c!}\left(\frac{\lambda}{\mu}\right)^c \frac{c\mu}{c\mu-\lambda} \tag{48}$$

Numărul de clienți care aşteaptă să fie serviți este

$$L_C = \sum_{n=0}^{\infty} n\pi_{c+n} = \frac{\pi_0}{c!}\left(\frac{\lambda}{\mu}\right)^c \sum_{n=0}^{\infty} n\left(\frac{\lambda}{c\mu}\right)^n = \pi_w \frac{\lambda}{c\mu-\lambda} \tag{49}$$

COMPARAȚIE: UN SISTEM CU DOUĂ COZI *(M/M/1)* ŞI O COADĂ *(M/M/2)*

Considerăm că avem două posturi de servire şi putem direcționa sosirile în aşa fel încât să existe două sisteme identice şi separate cu cozi de aşteptare sau să existe un singur sistem cu o singură coadă dar cu două posturi de servire.

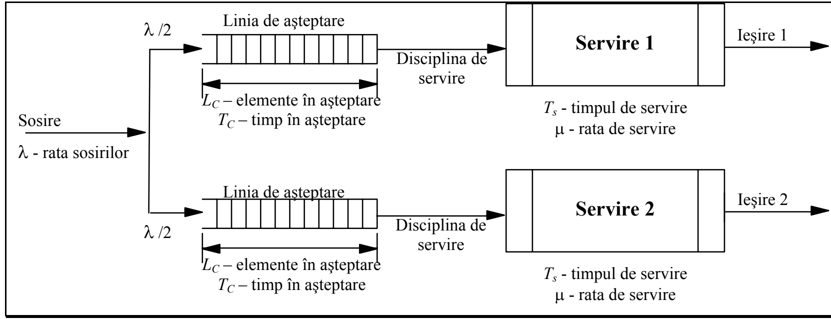


Fig. 109 Structura sistemului cu două cozi (M/M/1)

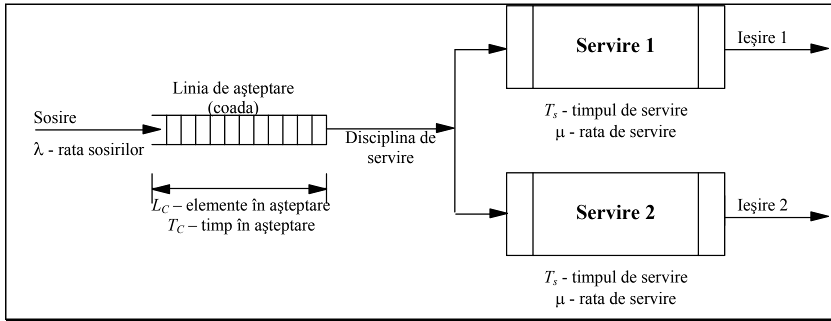


Fig. 110 Sistemul cu o coadă (M/M/2)

Pentru primul caz numărul mediu de clienți care aşteaptă să fie serviți în cele două cozi:

$$L_{C,2(M/M/1)} = 2L_C = 2\frac{\left(\lambda/2\right)^2}{\left(\mu-\lambda/2\right)\mu} = \left(\frac{\lambda}{\mu}\right)^2 \frac{1}{\left(2-\frac{\lambda}{\mu}\right)} \tag{50}$$

Pentru cazul cu şir de aşteptare unic şi două posturi de servire, conform (25) şi (26) relațiile între probabilități sunt:

$$\pi_0\lambda = \pi_1\mu\ ,\ \lambda\pi_1 = 2\mu\pi_2\ ,\ \lambda\pi_2 = 2\mu\pi_3\ ,\ \lambda\pi_3 = 2\mu\pi_4 ...,\ \lambda\pi_{n-1} = 2\mu\pi_n \tag{51}$$

Înlocuind iterativ corespunzător obținem:

$$\pi_1 = \pi_0 \frac{\lambda}{\mu},\ \pi_2 = \pi_1 \frac{\lambda}{2\mu} = \pi_0 \frac{\lambda}{\mu}\frac{\lambda}{2\mu} = \pi_0 \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2,$$

$$\pi_3 = \pi_2 \frac{\lambda}{2\mu} = \pi_0 \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2 \frac{\lambda}{2\mu} = \pi_0 \frac{1}{2}\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^3, \tag{52}$$

$$\pi_4 = \pi_3 \frac{\lambda}{2\mu} = \pi_0 \frac{1}{2}\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^3 \frac{\lambda}{2\mu} = \pi_0 \frac{1}{2^2}\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^4 \ldots\ \pi_n = \pi_0 \frac{1}{2^{n-2}}\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^n$$

Probabilitatea ca în sistem să nu fie nici un client este:

$$\pi_0 = \left[1 + \left(\frac{\lambda}{\mu}\right) + \left(\frac{\lambda}{\mu}\right)^2 \frac{\mu}{2\mu - \lambda}\right]^{-1} = \frac{2\mu - \lambda}{2\mu + \lambda} \tag{53}$$

Probabilitatea ca un nou client să nu fie servit este :

$$\pi_w = \frac{2\mu - \lambda}{2\mu + \lambda}\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2 \frac{2\mu}{2\mu - \lambda} = \left(\frac{\lambda}{\mu}\right)^2 \frac{\mu}{2\mu + \lambda} \tag{54}$$

Numărul mediu de entități în aşteptare este:

$$L_{C,(M/M/2)} = \pi_w \frac{\lambda}{2\mu - \lambda} = \left(\frac{\lambda}{\mu}\right)^2 \frac{\mu}{2\mu + \lambda}\frac{\lambda}{2\mu - \lambda} \tag{55}$$

Transformând corespunzător obținem:

$$L_{C,(M/M/2)} = (\lambda/\mu)^2 \frac{1}{(2 - \lambda/\mu)}\frac{\lambda/\mu}{(2 + \lambda/\mu)} = L_{C,2(M/M/1)} \frac{\lambda/\mu}{(2 + \lambda/\mu)} \tag{56}$$

Dar λ şi μ sunt pozitivi şi pentru oricare valori λ, μ

$$\frac{\lambda/\mu}{(2 + \lambda/\mu)} < 1 \tag{57}$$

Deci pentru oricare λ, μ este adevărată formula:

$$L_{C,(M/M/2)} < L_{C,2(M/M/1)} \tag{58}$$

Pentru verificare prezentăm comparativ în tabelul următor numărul mediu de clienți în aşteptare pentru diferite valori ale raportului λ/μ.

Tabelul 10. Comparare lungimilor cozilor

λ/μ	$L_{C,2(M/M/1)}$	$L_{C,(M/M/2)}$
1.99999	399996.00	199997.50
1.999	3996.00	1997.50
1.9	36.10	17.59
1.0	1.00	0.33
0.9	0.74	0.23
0.8	0.53	0.15
0.7	0.38	0.10
0.6	0.26	0.06
0.5	0.17	0.03
0.4	0.10	0.02
0.3	0.05	0.01
0.2	0.02	0.00
0.1	0.01	0.00

Concluzia ce se impune este că sistemele care au două posturi de servire ce lucrează simultan sunt mai eficiente dacă sunt configurate cu un fir unic de aşteptare decât dacă lucrează cu două cozi independente. Acelaşi lucru este valabil şi pentru cazul general al existenţei mai multor posturi de servire ($\forall c > 1$).

11.5. COZI DE AŞTEPTARE CU CAPACITATE FINITĂ

Până acum am studiat sisteme de servire în care capacitatea şirului de aşteptare era infinită. Evident asemenea situaţie este greu de susţinut în practică. În domeniul transporturilor, ca şi în majoritatea domeniilor, capacitatea constructivă a şirurilor de aşteptare este limitată. Solicitările de servire (clienţii) sosesc şi rămân în coada de aşteptare dacă există spaţiu, altfel solicitările se pierd. Numărul total de locuri din sistem include numărul de posturi de servire. În continuare vom prezenta câteva tipuri de cozi de aşteptare cu capacitate finită începând cu cea mai simplă: $(M/M/1/1)$.

SISTEME CU COZI DE AŞTEPTARE *(M/M/1/1)*

Acest tip de serviciu cu coadă de aşteptare constă dintr-un singur post de servire cu timpul de servire ce urmează o distribuţie exponenţială de rată μ în care sosesc cereri de servire ce urmează o distribuţie o distribuţie exponenţială cu rata λ. Sistemul nu are locuri suplimentare de aşteptare. Cererile care sosesc în perioada cât o servire este în curs se pierd.

Sistemul este foarte simplu de reprezentat: are doar două stări posibile, reprezentate în diagrama următoare:

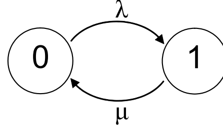


Fig. 111 *Diagrama tranziţiilor pentru (M/M/1/1)*

În stare staţionară

$$\pi_0 \lambda = \mu \pi_1 \tag{59}$$

şi

$$\pi_0 + \pi_1 = 1 \tag{60}$$

Înlocuind obţinem:

$$\pi_0 + \pi_0 \frac{\lambda}{\mu} = 1 \;\Rightarrow\; \pi_0 \left(1 + \frac{\lambda}{\mu}\right) = 1 \;\Rightarrow\; \pi_0 = \frac{\mu}{\mu + \lambda} \tag{61}$$

şi

$$\pi_1 = \pi_0 \frac{\lambda}{\mu} = \frac{\mu}{\mu + \lambda} \frac{\lambda}{\mu} = \frac{\lambda}{\mu + \lambda} \tag{62}$$

Deoarece μ şi λ sunt valori pozitive, indiferent de valoarea lor există o stare de echilibru.

Notă: Interesant este aici de observat că dacă μ şi λ sunt egale probabilitatea ca să existe un client în servire este $1/2$. Deci un asemenea serviciu porneşte din proiect cu 50% timp mort.

SISTEME CU COZI DE AŞTEPTARE *(M/M/1/N)*

Serviciul prezentat în secţiunea anterioară trebuie îmbunătăţit prin adăugarea de spaţiu de aşteptare. Se creează un număr suplimentar de $n-1$ locuri. Capacitatea totală a sistemului va fi n. Cererile de servire sosite rămân în coadă dacă este spaţiu sau se pierd. Comportamentul sistemului este figurat în diagrama următoare:

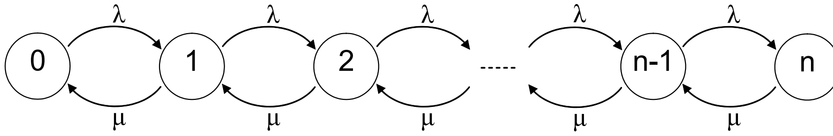


Fig. 112 Diagrama tranzițiilor pentru (M/M/1/N)

Aplicând ecuațiile (25) şi (26) rezultă:

$$\lambda\pi_0 = \mu\pi_1 \tag{63}$$

şi

$$\pi_i = \frac{\lambda}{\mu}\pi_{i-1} = \left(\frac{\lambda}{\mu}\right)^i \pi_0 \tag{64}$$

pentru $i = 1,2,3,4,...,N$

$$\sum_{i=0}^{N}\left[\left(\frac{\lambda}{\mu}\right)^i \pi_0\right] = 1 \;\Rightarrow\; \pi_0\sum_{i=0}^{N}\left(\frac{\lambda}{\mu}\right)^i = 1 \;\Rightarrow\; \pi_0\frac{1-(\lambda/\mu)^{N+1}}{1-(\lambda/\mu)} = 1 \tag{65}$$

Notând cu $a := \frac{\lambda}{\mu}$, dacă $a \neq 1$, probabilitatea ca în sistem să nu fie nici o cerere de servire este:

$$\pi_0 = \frac{1-a}{1-a^{N+1}} \tag{66}$$

şi pentru $i = 1,2,3,4,...,N$

$$\pi_i = \frac{(1-a)a^i}{1-a^{N+1}} \tag{67}$$

În cazul în care $a = 1$, probabilitatea ca în sistem să nu fie nici o cerere este:

$$\pi_0 = \frac{1}{N+1} \tag{68}$$

iar pentru $i = 1,2,3,4,...,N$

$$\pi_i = \frac{1}{N+1} \tag{69}$$

Probabilitatea ca un client să fie respins este π_N.

Numărul mediu de clienţi în sistem este dat de relaţia:

$$L = \sum_{i=0}^{N} i\pi_i = \sum_{i=0}^{N} i\frac{(1-a)a^i}{1-a^{N+1}} = \frac{a}{1-a} - \frac{(N+1)a^{N+1}}{1-a^{N+1}}, \quad a \neq 1 \tag{70}$$

şi

$$L = \sum_{i=0}^{N} i\pi_i = \sum_{i=0}^{N} \frac{i}{N+1} = \frac{N}{2}, \quad a = 1 \tag{71}$$

Dacă toate cele N locuri din sistem sunt ocupate atunci solicitările suplimentare sunt respinse. Deci rata intrărilor care sunt reţinute în sistem este:

$$\lambda_r = (1-\pi_N)\lambda = (1-\pi_0)\mu \tag{72}$$

În consecinţă, timpul mediu petrecut în sistem este:

$$T = L/\lambda_r \tag{73}$$

CARACTERISTICI GENERALE *(M/M/c/N)*

Acest tip de sistem este caracterizat prin sosiri care respectă o distribuţie exponenţială de rată λ, un număr de c puncte de servire cu timp servire cu distribuţie exponenţială de rată μ şi o capacitate de N locuri în sistem inclusiv în punctele de servire. Comportamentul lor este figurat în diagrama următoare:

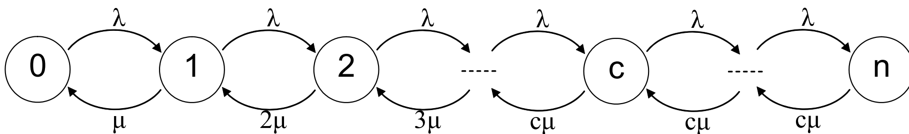


Fig. 113 Diagrama tranziţiilor pentru cozile de aşteptare (M/M/c/N)

Când se atinge echilibrul, probabilităţile pentru fiecare stare sunt:

$$\pi_i = \pi_0 \frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i \text{ pentru } i = 0,...,c \tag{74}$$

şi

$$\pi_i = \pi_0 \frac{1}{c!c^{i-c}}\left(\frac{\lambda}{\mu}\right)^i \text{ pentru } c \le i \le N \tag{75}$$

Notând $a := \lambda/(c\mu)$, ecuația (28) devine:

$$1 = \sum_{i=0}^{N}\pi_i = \pi_0\sum_{i=0}^{c-1}\frac{1}{i!}(ca)^i + \frac{\pi_0}{c!}(ca)^c\sum_{i=0}^{N-c}a^i = \pi_0\left(\sum_{i=0}^{c-1}\frac{1}{i!}(ca)^i + \frac{(ca)^c}{c!}\sum_{i=0}^{N-c}a^i\right) \tag{76}$$

Probabilitatea ca în sistem să nu fie nici un client este:

$$\pi_0 = \left[\sum_{i=0}^{c-1}\frac{1}{i!}(ca)^i + \frac{(ca)^c}{c!}\sum_{i=0}^{N-c}a^i\right]^{-1} \tag{77}$$

Numărul mediu de solicitări care aşteaptă să fie servite este dat de:

$$\begin{aligned} L_c &= \sum_{i=c+1}^{N}(i-c)\pi_i = \frac{\pi_0(ca)^c}{c!}\sum_{i=0}^{N-c}ia^i \\ &= \frac{\pi_0(ca)^c a}{c!}\frac{1-(N-c+1)a^{N-c}+(N-c)a^{N-c+1}}{(1-a)^2} \end{aligned} \tag{78}$$

COZI DE AŞTEPTARE *(M/M/c/c)*

Cozilor de tipul $(M/M/c/c)$ se caracterizează prin existența unui număr de locuri în sistem egal cu numărul de posturi de servire (nu există locuri suplimentare). Niciodată un client acceptat nu va aştepta până să fie servit. O cerere intră direct în procesare sau nu este admisă de loc. Comportamentul acestui tip de sistem este figurat în diagrama următoare:

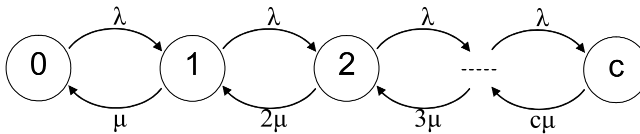


Fig. 114 Diagrama tranzițiilor pentru cozile de aşteptare (M/M/c/c)

Când se atinge echilibrul, probabilitățile pentru fiecare stare sunt:

$$\pi_i = \pi_0 \frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i \text{ pentru } i = 0,\dots,c \tag{79}$$

Notând $a := \frac{\lambda}{\mu}$, ecuația (28) devine:

$$1 = \sum_{i=0}^{c} \pi_i = \pi_0 \sum_{i=0}^{c} \frac{a^i}{i!} \tag{80}$$

Probabilitatea ca în sistem să nu fie nici-un client este:

$$\pi_0 = \left[\sum_{i=0}^{c} a^i / i! \right]^{-1} \tag{81}$$

De interes este în acest caz probabilitatea ca cererile de servire să fie pierdute. Aceasta se întâmplă când toate cele *c* locațiile sunt ocupate:

$$\pi_c = \frac{a^c / c!}{\left[\sum_{i=0}^{c} a^i / i! \right]} \tag{82}$$

Aceasta este formula de pierdere a lui Erlang, dedusă de A.K. Erlang în 1917. Aceasta înseamnă că π_c din solicitări vor şi pierdute. Este interesant de remarcat faptul că formula de pierdere este adevărată indiferent de distribuția servirilor.

Pentru a ca sistemul să fie eficient este necesar ca π_c să aibă o valoare sub un prag considerat.

11.6. COZILE DE AŞTEPTARE DE TIPUL *(M/G/1)*

Acestea sunt cozi în care sosirile sunt procese Poisson cu rata $\lambda > 0$ şi în care timpii de servire a cererilor sunt independente cu aceeaşi funcție cumulată a distribuției $G(t)$. În termeni probabilistici spunem că dacă s_i şi s_j sunt timpii de servire a cererii *i* şi *j* unde $i \neq j$ atunci: s_i şi s_j variabile aleatoare independente; şi $G(t) = P(s_i \leq t) = P(s_j \leq t), \forall t \geq 0$.

Media timpului de servire este $E[S_i] = E[S_j] = E[B] = 1/\mu$ unde μ este rata servirii. Abaterea standard a timpilor de servire este $E[(s_i - E[B])^2] = \sigma_B$. Ordinea de tratare a cererilor se presupune a fi primul venit primul servit. Raportul λ/μ se notează cu ρ.

Numărul de cereri din sistem nu se mai supune regulilor procesului Markov. Chiar dacă nu mai putem beneficia de avantajele acestui tip proces, totuşi

caracteristicile acestui tip de coadă (lungime, timp de aşteptare, timp în sistem) pot fi deduse. Mai întâi ca calculăm $\overline{T_c}$ timpul mediu de aşteptare pentru a fi servit. Acesta este timpul pe care un „client” îl petrece în „sala de aşteptare”. Acesta este timpul mediu de aşteptare pentru încheierea sarcinii în curs la care se adună suma timpilor de servire a cererilor care aşteptă.

$$\overline{T_c} = \overline{T_R} + \overline{N_c} \cdot E[B] \tag{83}$$

Unde :

- $\overline{T_R}$ timpul mediu de aşteptare pentru terminarea pentru încheierea procesării în curs.
- $\overline{N_c}$ numărul mediu de cereri care aşteaptă;
- $E[B]$ timpul mediu de servire;

T_R este timpul pe care îl are de aşteptat cel care intră în coada de aşteptare pentru a se încheia de procesat cererea în curs. Dacă în sistem nu este nimeni atunci $T_R = 0$.

Conform legii lui Little media lungimii cozii este:

$$\overline{N_c} = \lambda \overline{T_c} \tag{84}$$

Înlocuind obţinem:

$$\overline{T_c} = \overline{T_R} + \lambda \overline{T_c} \cdot E[B] \Rightarrow \overline{T_c} - \frac{\lambda}{\mu}\overline{T_c} = \overline{T_R} \Rightarrow \overline{T_c} = \frac{\overline{T_R}}{1-\rho} \tag{85}$$

Acum trebuie să calculăm valoare $\overline{T_R}$. Pentru o mai bună înţelegere să reprezentăm timpul rămas pentru fiecare dintre cererile de servire.

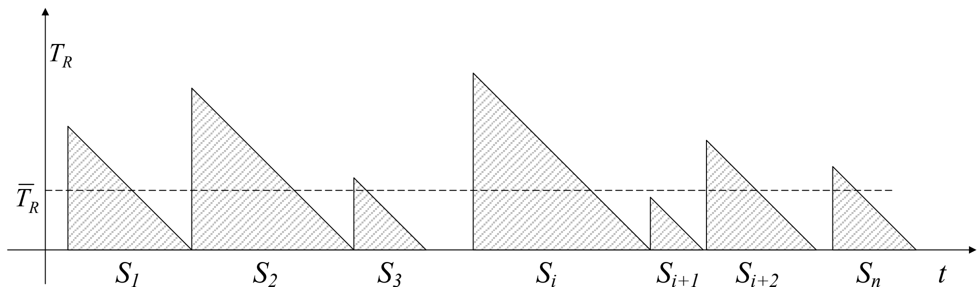


Fig. 115 Timpul rămas până la sfârşitul servirii

Fiecare dintre triunghiurile din figură este drept şi isoscel. Numărul de triunghiuri este dat de rata sosirilor $\overline{n} = \lambda \overline{T}_R$.

$$\overline{T}_R = \frac{1}{t}\int_0^t T_R(\tau)d\tau = \frac{1}{t}\sum_{i=1}^{n}\left(\frac{1}{2}s_i^2\right) = \frac{1}{2}\frac{n}{t}\frac{1}{n}\sum_{i=1}^{n}s_i^2 \tag{86}$$

La limită, pentru starea staţionară, $\overline{T}_R$ ia forma:

$$\overline{T}_R = \frac{1}{2}\lambda E\left[S^2\right] \tag{87}$$

Revenind la timpul de aşteptare în coadă pentru a fi servit obţinem:

$$\overline{T}_c = \frac{\lambda E\left[S^2\right]}{2(1-\rho)} \tag{88}$$

Aceasta este formula Pollaczek-Hincin pentru medie. Calculând $E\left[S^2\right]$ rezultă:

$$\overline{T}_c = \frac{\lambda\left[(E[B])^2 + \sigma_B^2\right]}{2(1-\rho)} = \frac{\lambda\left[(1/\mu)^2 + \sigma_B^2\right]}{2(1-\rho)} \tag{89}$$

Timpul total de aşteptare este:

$$\overline{T} = \overline{T}_c + E[B] = \frac{\lambda\left[(1/\mu)^2 + \sigma_B^2\right]}{2(1-\rho)} + \frac{1}{\mu} \tag{90}$$

Corespunzător se calculează numărul de cereri din coadă:

$$\overline{N}_c = \lambda \overline{T}_c = \frac{\lambda^2\left[(1/\mu)^2 + \sigma_B^2\right]}{2(1-\rho)} \tag{91}$$

şi respectiv:

$$\overline{N} = \lambda \overline{T} = \frac{\lambda^2\left[(1/\mu)^2 + \sigma_B^2\right]}{2(1-\rho)} + \rho \tag{92}$$

Aceste formule sunt suficient de generale prin condiţiile impuse.

Pentru verificare să considerăm cazul $(M / M / 1)$ unde $E[B] = \sigma_B$:

$$\overline{T_c} = \frac{\lambda\left[(E[B])^2 + \sigma_B^2\right]}{2(1-\rho)} = \frac{\lambda\left[(1/\mu)^2 + (1/\mu)^2\right]}{2(1-\rho)} = \frac{\lambda}{(\mu-\lambda)\mu} \tag{93}$$

şi

$$\overline{T} = \frac{\rho}{(1-\rho)\mu} + \frac{1}{\mu} = \frac{\rho+1-\rho}{(1-\rho)\mu} = \frac{1}{(1-\rho)\mu} = \frac{1}{(\mu+\lambda)} \tag{94}$$

Valorile obținute sunt identice cu cele calculate la subcapitolul anterior corespunzător.

11.7. ANALIZA COZILOR DE TIPUL *(M/G/-/-/N)*

O clasă de cozi care poate fi remarcată este $(M/G/-/-/N)$ care este caracterizată prin faptul că populația care generează intrările este limitată. Aceasta înseamnă că pe măsură ce în coada de aşteptare se acumulează cereri sau clienți, în exterior rămân mai puține entități care să genereze cereri de servire. Rata intrărilor va scădea proporțional cu scăderea populației externe. Deci rata intrărilor nu mai este fixă, ca la situațiile precedente, ci variabilă după o anumită lege.

Pentru exemplificare vom considera cazul sistemelor cu coadă de aşteptare $(M/M/1//N)$.

Acest tip de coadă de aşteptare este caracterizat de prezența unui singur post de servire cu distribuția timpului de servire de tip exponențial cu parametrul μ, numărul de locuri în coadă este „infinit”, sosirea se face aleator cu distribuția timpilor de sosire de tip exponențial. Populația externă este N, de aceea sunt în fapt necesare doar N locuri în coada de aşteptare. Elementele populației considerate sunt identice şi pot genera cereri de servire cu o rată λ_e. Având o populație p, rata intrărilor va fi $p\lambda_e$. Comportamentul sistemului este figurat schematic în diagrama următoare:

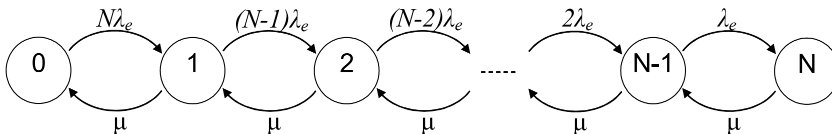


Fig. 116 Diagrama tranzițiilor pentru (M/M/1/N)

Aplicând ecuațiile (25) şi (26) rezultă:

$$\pi_0 N\lambda_e = \mu\pi_1 \Rightarrow \pi_1 = \frac{\pi_0 N\lambda_e}{\mu} \tag{95}$$

şi

$$\pi_i = \frac{(N-i+1)\lambda_e}{\mu}\pi_{i-1} = \frac{N!\lambda_e^i}{i!\mu^i}\pi_0 \tag{96}$$

pentru $i = 1,2,3,4,...,N$. Relaţia se verifică de fapt şi pentru $i = 0$. De aceea putem scrie:

$$\sum_{i=0}^{N}\pi_i = \sum_{i=0}^{N}\left[\frac{N!}{i!}\left(\frac{\lambda_e}{\mu}\right)^i \pi_0\right] = 1 \tag{97}$$

Notând cu $a := \frac{\lambda_e}{\mu}$, obţinem:

$$N!\pi_0\sum_{i=0}^{N}\left[\frac{a^i}{i!}\right] = 1 \tag{98}$$

Adică probabilitatea ca în sistem să nu fie nici o cerere este

$$\pi_0 = \left[N!\pi_0\sum_{i=0}^{N}\left(\frac{a^i}{i!}\right)\right]^{-1} \tag{99}$$

Celelalte probabilităţi se calculează aplicând corespunzător ecuaţiile (96).

11.8. COZI DE AŞTEPTARE CU PRIORITATE

NOŢIUNI GENERALE

Nu totdeauna cererile care aşteaptă servirea într-o coadă sunt egale din punctul de vedere al celui care face servirea. Unii clienţi au prioritate în faţa altora. Prioritatea poate fi continuă sau discretă (se împart cererile în câteva clase de prioritate şi se face departajarea între ei). Această prioritate poate fi stabilită la intrare în coadă şi rămâne fixă pe durata aşteptării sau se poate modifica pe parcurs.

Exemple de cereri de servire cu prioritate fixă sunt vehiculele poliţiei, salvării, pompierilor care pot trece pe culoarea roşie a semaforului în timp ce celelalte stau în coada de aşteptare formată.

Un exemplu de prioritate ce se modifică pe parcurs este dat de coada de aşteptare la ANL[1] pentru construirea de case pentru tineri. Prioritatea este dată de o

[1] Agenţia Naţională a Locuinţei

serie de factori legaţi de situaţia familială, de venituri dar şi de timpul petrecut în coada de aşteptare. Deoarece servirea în acest caz se face pe loturi, de fiecare dată când se face o servire se reevaluează prioritatea.

Pentru cazul în care se asociază o prioritate fixă, cererile care au aceeaşi prioritate sunt servite într-o ordine dintre cele descrise în sub-capitolul introductiv (de obicei ordinea este primul sosit primul servit). În general se asociază un indice de prioritate. Se face convenţia că un indice de valoare mai mică reprezintă o prioritate mai mare.

Sistemele cu cozi de aşteptare cu priorităţi pot reacţiona diferit la o cerere prioritară când în curs de servire se află o cerere cu o prioritate mai scăzută. Aceste cereri pot să aştepte sau pot bloca servirea în curs luându-i locul.

Funcţie de modul de tratare a cererilor sistemele cu cozi de aşteptare cu priorităţi pot fi:

- ***fără întrerupere*** (eng. *non-preemptive*) – cererile cu o prioritate mai scăzută aflate în procesul de servire nu sunt întrerupte de sosirea unei cereri mai prioritare;
- ***cu întrerupere*** (eng. *preemptive*) – cererile aflate în servire sunt întrerupte pentru a face loc servirii unei cereri de prioritate mai mare.
  - ***cu întrerupere reluată*** (eng. *preemptive resume*) – cererile întrerupte sunt reluate exact de unde au fost lăsate şi operaţia se continuă până la închiere sau la sosirea unei alte cereri prioritare.
  - ***cu întrerupere repetată*** (eng. *preemptive repeat*) – cererile întrerupte sunt reluate de la început, operaţiile deja efectuate se pierd.

COZI DE AŞTEPTARE CU PRORITATE ŞI ÎNTRERUPERE

Acest tip de coadă de aşteptare va procesa întotdeauna o cerere cu cea mai înaltă prioritate. Pentru rezolvarea cozilor de tip $(M/M/-/-)$ cu prioritate vom trasa diagrama tranziţiilor (este vorba de procese Poisson) şi se vor trata sistemele de ecuaţie rezultate obţinând probabilităţile stărilor.

Vom începe analiza prin prezentarea cozilor de tip $(M/M/1)$ cu prioritate.

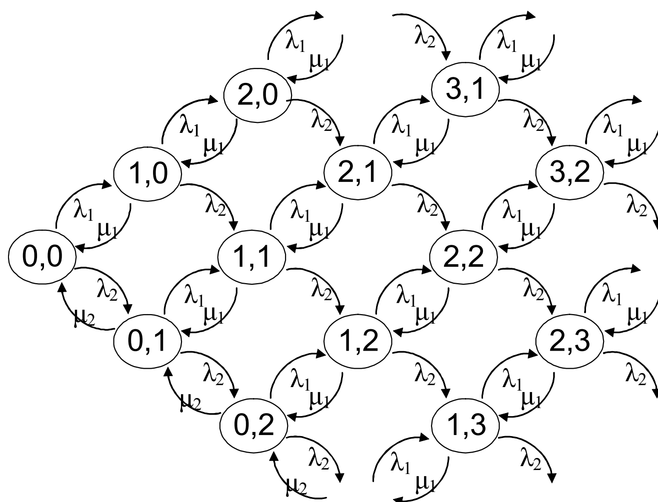


Fig. 117 Diagrama tranziţiilor pentru cozile de aşteptare (M/M/1) cu două niveluri de prioritate

Pentru rezolvarea acestui tip de sistem trebuie să scriem ecuaţiile de echilibru în fiecare nod.

Ca un exemplu vom considera sistemul de tip $(M/M/1/3)$ care are două niveluri de prioritate. Acest tip de coadă de aşteptare ar putea fi un model pentru un atelier de reparaţii care are doar trei locuri inclusiv cel de pe rampă. Sarcina sa este de a repara prioritar maşinile proprii ale organizaţiei care patronează atelierul (ex. o antrepriză de lucrări drumuri şi poduri), dar din motive de autofinanţare acceptă cereri de la populaţie. Regulamentul spune că dacă o maşină a organizaţiei solicită o reparaţie şi există locuri libere, reparaţia unei maşini externe este întreruptă şi maşina proprie este reparată. Maşinile de aceeaşi prioritate se tratează după disciplina primul sosit primul servit. Cererile deja acceptate (maşinile externe au plătit deja avans) nu pot fi eliminate. Dacă toate locurile sunt ocupate atunci cererea este directată spre alt atelier.

Pentru cazul $(M/M/1/3)$ cu două niveluri de prioritate diagrama tranziţiilor are următoarea formă:

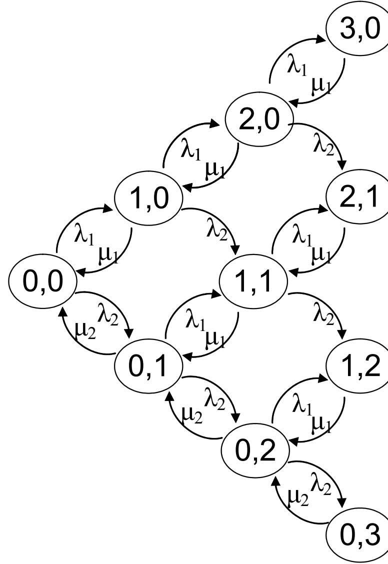


Fig. 118 Diagrama tranzițiilor pentru cozile de aşteptare (M/M/1/3) cu două niveluri de prioritate

Analizând mulțimea stărilor posibile şi considerând echilibrul în noduri, pentru starea staționară putem scrie:

$$\begin{aligned}
\pi_{0,0}(\lambda_1+\lambda_2) &= \pi_{1,0}\mu_1+\pi_{0,1}\mu_2,\\
\pi_{1,0}(\lambda_1+\lambda_2+\mu_1) &= \pi_{0,0}\lambda_1+\pi_{2,0}\mu_1,\\
\pi_{0,1}(\lambda_1+\lambda_2+\mu_2) &= \pi_{0,0}\lambda_2+\pi_{1,1}\mu_1+\pi_{0,2}\mu_2,\\
\pi_{1,1}(\lambda_1+\lambda_2+\mu_1) &= \pi_{0,1}\lambda_1+\pi_{1,0}\lambda_2+\pi_{2,1}\mu_1,\\
\pi_{2,0}(\lambda_1+\lambda_2+\mu_1) &= \pi_{1,0}\lambda_1+\pi_{3,0}\mu_1,\\
\pi_{0,2}(\lambda_1+\lambda_2+\mu_2) &= \pi_{0,1}\lambda_2+\pi_{1,2}\mu_1+\pi_{0,3}\mu_2,\\
\pi_{2,1}\mu_1 &= \pi_{1,1}\lambda_1+\pi_{2,0}\lambda_2,\\
\pi_{1,2}\mu_1 &= \pi_{1,1}\lambda_2+\pi_{0,2}\lambda_1,\\
\pi_{3,0}\mu_1 &= \pi_{2,0}\lambda_1,\\
\pi_{0,3}\mu_2 &= \pi_{0,2}\lambda_2
\end{aligned} \tag{100}$$

La ecuațiile anterioare se adaugă:

$$\pi_{0,0}+\pi_{1,0}+\pi_{0,1}+\pi_{1,1}+\pi_{2,0}+\pi_{0,2}+\pi_{2,1}+\pi_{1,2}+\pi_{3,0}+\pi_{0,3}=1 \tag{101}$$

Din rezolvarea ecuațiilor se obține distribuția probabilităților.

11.9. REŢELE CU COZI DE AŞTEPTARE

Cazurile studiate anterior sunt doar situaţii în care avem sisteme cu singură coadă de aşteptare izolată. În practică însă, aceste sisteme sunt conectate între ele şi formează ***reţele de cozi de aşteptare*** care trebuie considerate şi studiate în complexitatea lor. O cerere de servire aşteaptă într-o coadă, este rezolvată după care trece în alta şi aşa mai departe. În final un client poate să iasă din sistem sau poate să reia procesul de la început.

O reţea cu cozi de aşteptare poate fi ***deschisă*** sau ***închisă***.

Într-o reţea deschisă, cererea / clientul vine din exteriorul sistemului, parcurge cozile în totalitate sau în parte şi părăseşte sistemul. De obicei, în acest caz, se consideră că populaţia de cereri / clienţi este infinită. Sistemul poate conţine conexiuni inverse şi puncte de despărţire. Cozile de aşteptare pot avea stabilite priorităţi.

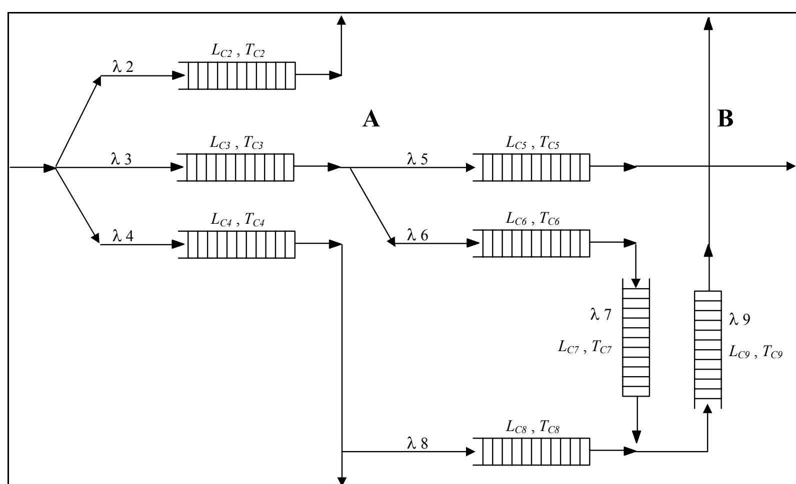


Fig. 119 Exemplu schematic de reţea deschisă cu cozi de aşteptate

În figura de mai sus se poate vedea modelul unui sens de circulaţie pe un bulevard. Autovehiculele intră în sistem, parcurg primul segment intră în intersecţia **A**, aşteaptă la semafor, trec şi intră in intersecţia **B** unde, din nou, trebuie să aştepte la semafor. Din anumite considerente în a doua intersecţie este interzisă întoarcerea la stânga. De aceea, în intersecţia anterioară (**A**) trebuie să ocolim pe o stradă laterală şi să venim în intersecţia **B** perpendicular pe bulevard.

Fiecare dintre fluxurile de vehicule este modelat prin cozi de aşteptare separate şi puncte în care traficul se împarte pe direcţii diferite.

Reţeaua închisă presupune că întreaga populaţie care poate genera cereri de servire este cuprinsă în interiorul sistemului.

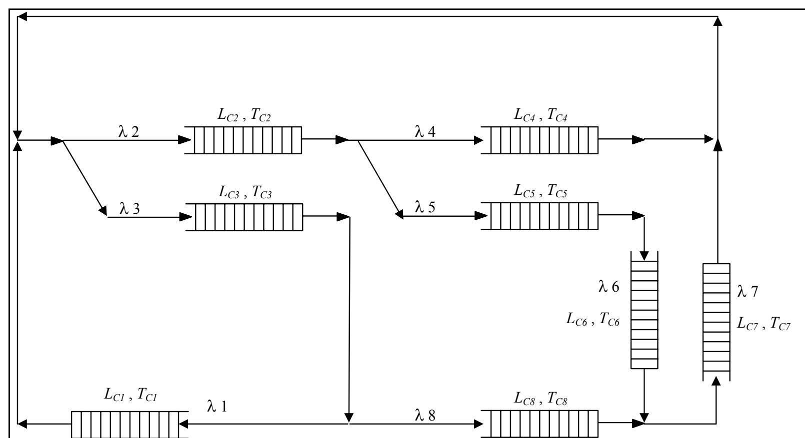


Fig. 120 Exemplu schematic de rețea închisă cu cozi de aşteptate

În practică, traficul trece prin mai mult de un nod. Ca o consecință, vehiculele trebuie să aştepte în câteva cozi individuale. Un set de cozi conectate se studiază ca rețea de cozi de aşteptare. Rețelele pot fi studiate sub diferite presupuneri privind distribuția sosirilor, distribuția servirii, disciplina servirii, dimensiunea zonei de aşteptare etc. (Balsamo et al. 2001 [181]).

Recensămintele de trafic anterioare ne pot indica numărul de vehicule pe intervale de timp pentru fiecare zi a săptămânii. Acestea ar fi sosirile. Pe de altă parte fiecare sector rutier are o anumită capacitate maximă. O analiză preliminară poate porni de presupunerea că fluxul de vehicule şi procesul de trecere sunt deterministice. Nu este o realitate ci doar o aproximare convenabilă utilizată în procesul de modelare.

Pentru a avea rezultate cât mai corecte ar trebui să presupunem o rețea deschisă cu cozi de aşteptare $(G/G/1)$ (Vandaele et al. 1999 [218]). Modelul $(M/M/1)$ a fost intens utilizat din cauza simplității, dar nu este util fiind departe de realitate. Pentru benzi multiple Jain şi Smith au dezvoltat un model $(G/G/k)$ [195]. În condiții de trafic necongestionat, Van Woense şi Vandaele au demonstrat că cel mai bun model ce se poate obține este $(M/G/1)$ [216].

11.10. BIBLIOGRAFIE

[177] ***: *Analysis of Some Queuing Models in Real-Time Systems* (2nd Ed) - IBM Technical Publications Dept. 1971.
[178] Abramowitz M., I.A. Stegun: *Handbook of mathematical functions*, Dover, 1965.
[179] Adan I., Y.Zhao: *Analyzing GI/Er/1 queues*; Opns. Res. Lett., 19 (1996), pp. 183-190.
[180] Adan I.J.B.F., W.A. van de Waarsenburg, J. Wessels: *Analyzing Ek/Er/c queues*; EJOR, 92, pp.112-124, 1996.
[181] Balsamo, S. de Nitto Personé, V., R. Onvural: *Analysis of Queueing Networks with Blocking*; Kluwer Academic Publishers, 2001.

[182] Bruijn N.G. de: *Asymptotic methods*; Dover, 1981.
[183] Bunday B.D.: *An introduction to queueing theory*; Arnold, London, 1996.
[184] Buzacott J.A., J.G. Shanthikumar: *Stochastic models of manufacturing systems*; Prentice Hall, Englewood Cliffs, 1993.
[185] Cohen J.W.: *On regenerative processes in queueing theory*; Springer, Berlin, 1976.
[186] Cohen J.W.: *The single server queue*; North-Holland, Amsterdam, 1982.
[187] Cooper Robert B.: *Introduction to Queueing Theory*; Second edition, Elsevier North Holland Inc., New York, 1981. Fourth Edition, Ceep Press Books, 1990
[188] Daganzo, C.F.: *Fundamentals of Transportation and Traffic Operations*; Elsevier Science Ltd., Oxford, 1997.
[189] Dshalalow J.H. (editor): *Advances in Queueing: Theory, Methods and Open Problems*; CRC Press, Boca Raton, 1995.
[190] Gross Donald, Harris Carl M.: *Fundamentals of queueing theory*; Wiley, Chichester, 1985. 1998
[191] Heidemann, D.: *A queueing theory approach to speed-flow-density relationships, Transportation and Traffic Theory*; Proceedings of the 13th International Symposium on Transportation and Traffic Theory, Lyon, France, 14-26 July 1996.
[192] Heijden M.C. van der: *Performance analysis for reliability and inventory models*; Thesis, Vrije Universiteit, Amsterdam, 1993.
[193] Heyman D.P., M.J. Sobel: *Stochastic models in operations research*; McGraw-Hill, London, 1982.
[194] Hillier F. S., Lieberman G. J.: *Introduction to Operations Research*; (6th Ed) 1995.
[195] Jain, R., Smith J.M.: *Modelling vehicular traffic flow using M/G/C/C state dependent queueing models*; Transportation Science, 31, pp.324-336, 1997.
[196] Johnson M.A.: *An emperical study of queueing approximations based on phase-type approximations*; Stochastic Models, 9 (1993), pp.531-561.
[197] Kendall DG: *Some Problems in the Theory of Queues*; Jurnal of the Royal Statistical Society, B13, pp.151-185, 1951.
[198] Kendall DG: *Stochastic Processes Occurring in the Theory of Queues and Their Analysis by Means of the Imbadded Markov Chaine*; The Annals of Mathematical Statistics 24, pp.338-354, 1953.
[199] Kleinrock Leonard.: *Queueing Systems, Volume I: Theory*; Wiley, New York, 1975.
[200] Kleinrock Leonard: *Queueing Systems - Volume II: Computer Applications*; Wiley-InterScience, 1976
[201] Lee A.M.: *Applied queuing theory*; MacMillan, London, 1968.
[202] Little J.D.C.: *A Proof of the Queuing Formula:* $N = \lambda T$; Operations Research 9, No.3, pp.383-387, 1961.
[203] Marie R.A.: *Calculating equilibrium probabilities for* $\lambda(n)/C_k/1/N$ *queue*; in: Proceedings Performance' 80, Toronto, (May 28-30, 1980), pp.117-125.
[204] Newell G.F.: *Applications Of Queueing Theory*; Chapman and Hall, London 1971. Second Edition, 1982.
[205] Page E.: *Queueing Theory*; in Operation Research, 1972
[206] Ross S.M.: *Introduction to probability models*; 6th ed., Academic Press, London, 1997.
[207] Rubinovitch M.: *The slow server problem: a queue with stalling*; J. Appl. Prob., 22, pp. 879-892, 1985.
[208] Rubinovitch M.: *The slow server problem*; J. Appl. Prob., 22, pp. 205-213, 1985.

[209] Schassberger R.S.: *On the waiting time in the queueing system GI/G/1*; Ann. Math. Statist., 41, pp. 182-187, 1970.

[210] Schassberger R.S.: *Warteschlangen*; Springer-Verlag, Berlin, 1973.

[211] Stidham S.: *A last word on* $L = \lambda W$; Opns. Res., 22, pp. 417-421, 1974.

[212] Takacs L.: *Introduction to the theory of queues*; Oxford, 1962.

[213] Tijms H.C.: *Stochastic modelling and analysis: a computational approach*; John Wiley & Sons, Chichester, 1990.

[214] Tijms H.C.: *Stochastic models: an algorithmic approach*; John Wiley & Sons, Chichester, 1994.

[215] Turbuț Gh. et al.: *Inginerie de sistem, automatizări şi informatică în transporturi*; Editura Tehnică, Bucureşti 1988.

[216] Van Woensel, T. and N. Vandaele: *Empirical validiation of a queueing approach to uninterrupted traffic flows*; UFSIA research paper , 2002.

[217] Van Woensel, T., R. Creten and N. Vandaele: *Managing the environmental externalities of traffic logistics: the issue of emissions*; POM journal special issue, vol. 10, nr.2, pp.207-223, 2001.

[218] Vandaele N. L. De Boeck and D. Callewier: *An Open Queueing Network for Lead Time Analysis*; UFSIA research paper, 99-017, 1999.

[219] Vandaele, N., Van Woensel T. and A. Verbruggen: *A queueing based traffic flow model*; Transportation Research-D: Transport and environment, vol. 5 nr. 2, pp. 121-135, 2000.

[220] Walrand J.: *An Introduction to Queueing Networks*; 1988.

[221] Whitt W.: *Approximating a point process by a renewal process I: two basic methods*; Opns. Res., 30, pp. 125-147, 1986.

[222] Whittaker E.T., G.N. Watson: *Modern analysis*; Cambridge, 1946.

[223] Wolff R.W.: *Poisson arrivals see time averages*; Opns. Res., 30, pp. 223-231, 1982.

[224] Wolff R.W.: *Stochastic modeling and the theory of queues*; Prentice-Hall, London, 1989.

ISBN 973-86343-3-4



COLECȚIA INGINERIA PODURILOR